Probabilistic State Translation in Extensive Games with Large Action Sets

David Schnizlein **Michael Bowling** **Duane Szafron**
{schnizle, bowling, duane}@cs.ualberta.ca
Department of Computing Science
University of Alberta
Edmonton, AB, Canada T6G2E8

Abstract

Equilibrium or near-equilibrium solutions to very large extensive form games are often computed by using abstractions to reduce the game size. A common abstraction technique for games with a large number of available actions is to restrict the number of legal actions in every state. This method has been used to discover equilibrium solutions for the game of no-limit heads-up Texas Hold'em. When using a solution to an abstracted game to play one side in the un-abstracted (real) game, the real opponent actions may not correspond to actions in the abstracted game. The most popular method for handling this situation is to translate opponent actions in the real game to the closest legal actions in the abstracted game. We show that this approach can result in a very exploitable player and propose an alternative solution. We use probabilistic mapping to translate a real action into a probability distribution over actions, whose weights are determined by a similarity metric. We show that this approach significantly reduces the exploitability when using an abstract solution in the real game.

1 Introduction

Many complex problems involving multiple agents can be solved using an extensive form game tree formulation. However, some problems are too complicated to solve, since the resulting extensive game tree is too large. One method for solving problems whose extensive game trees are too large is to create an abstraction of the game that results in a smaller game tree and solve this abstract game instead of the real game. The abstraction approach creates two problems. First, there is information lost during the abstraction process. An equilibrium solution to the abstract game is not an equilibrium solution to the real game. When the real game is much larger than the abstract game, the respective equilibrium solutions may be dissimilar enough that the abstract solution is actually a poor strategy in the real game. Second, to use the solution of the abstract game to play the real game one must create a mapping between the states of the real game and the states of the abstract game. This mapping can become very complex as the difference in size of the abstract and real games increases.

Extensive games have been used to try to create agents for several variants of poker. Poker has been thoroughly studied for some time [Billings *et al.*, 2002] and has grown in popularity in recent years. In addition, three annual AAAI Computer Poker Competitions [Zinkevich and Littman, 2006] have helped spur poker research and have resulted in new algorithms for solving extensive games. Two such algorithms are regret minimization [Zinkevich *et al.*, 2008] and gradient-based algorithms [Gilpin *et al.*, 2007].

Most of the literature on poker agents describes the problem of two-player (heads-up) limit Texas Hold'em, whose size is approximately 10^{18}. Recent algorithms have only been able to solve games whose size is about 10^{12} [Zinkevich *et al.*, 2008; Gilpin *et al.*, 2007], and abstractions have been used to bridge this size gap. In contrast, the size of the no-limit version of two player Texas Hold'em is approximately 10^{71} [Gilpin *et al.*, 2008]. The reason this game is so much larger is that there are many more possible actions in every state. In the limit version, each player has at most only three actions: fold, call or raise a fixed number of chips. In the no-limit version, each player can have hundreds of actions, since a player can raise any amount from the size of the last bet to the player's entire stack. Even though the no-limit game is much larger, the abstractions being used are of the same size as the ones being used in the limit game. The effect of trying to use an abstract solution to play a real game that is more than 10^{50} times larger has not been studied very carefully.

In this paper, we investigate the effect of using solutions obtained by abstracting extensive form games with large action sets. We formalize the concept of *state translation*, the process of translating a state in the real game to a state in the abstract game, and show how translation can be separated from the abstraction process to provide the agent with more flexibility. Additionally, we formalize the current methods of translation in poker and suggest a new method that is generalizable to any extensive game. Finally, we show that the current translation methods used in poker create an extremely exploitable agent and that our new translation method reduces this exploitability.

2 Background

2.1 Extensive Games

An extensive game involves combinations of actions taken by players and chance. For example, in poker, the actions would be the player actions (fold, call or raise) together with the cards dealt (chance). Each list of actions is called a history and hidden information can be modeled by partitioning the histories into sets, called information sets, whose elements cannot be distinguished from one another by an individual player. For example, in poker, two histories that differ only by the opponent's cards would be indistinguishable by a player and be in the same information set.

Formally, we can define an extensive game as follows.

Definition 1 (Extensive Game) *[Osborne and Rubenstein, 1994, p. 200] A finite extensive game with imperfect information is denoted Γ and has the following components:*

- *A finite set N of **players**.*
- *A finite set H of sequences, the possible **histories** of actions, such that the empty sequence is in H and every prefix of a sequence in H is also in H. $Z \subseteq H$ are the **terminal histories**. No sequence in Z is a strict prefix of any sequence in H. $A(h) = \{a : (h, a) \in H\}$ are the actions available after a non-terminal history $h \in H \backslash Z$.*
- *A **player function** P that assigns to each non-terminal history a member of $N \cup \{c\}$, where c represents chance. $P(h)$ is the player who takes an action after the history h. If $P(h) = c$, then chance determines the action taken after history h. Let H_i be the set of histories where player i chooses the next action.*
- *A function f_c that associates with every history h for which $P(h) = c$ a probability measure $f_c(\cdot|h)$ on $A(h)$. $f_c(a|h)$ is the probability that a occurs given h, where each such probability measure is independent of every other such measure.*
- *For each player $i \in N$, a partition $\mathbf{I}_i$ of H_i with the property that $A(h) = A(h')$ whenever h and h' are in the same member of the partition. $\mathbf{I}_i$ is the **information partition** of player i; a set $I_i \in \mathbf{I}_i$ is an **information set** of player i.*
- *For each player $i \in N$, a **utility function** u_i that assigns each terminal history a real value. $u_i(z)$ is rewarded to player i for reaching terminal history z. If $N = \{1, 2\}$ and for all z, $u_1(z) = -u_2(z)$, an extensive form game is said to be **zero-sum**.*

A **strategy** σ for a game is a weighted set of legal actions for every history. A **best response** for player i to strategy σ is the strategy that maximizes its utility assuming all other players play according to σ.

2.2 Game Abstraction

Large games can be abstracted by increasing the size of information sets to reduce their total number. Since an information set contains histories and a history is a sequence of player and chance actions, there are two techniques for increasing the size of an information set. The first technique combines chance actions together into *buckets*. For example, in poker, multiple player hands could be combined together into a single bucket. The second technique artificially reduces the number of allowable player actions in the abstraction. For example, in no-limit poker, a raise could artificially be constrained to be the amount currently in the pot (*pot*) or the current player's full stack (*all-in*).

More formally, game abstraction is defined as follows.

Definition 2 (Abstraction) *[Waugh* et al.*, 2009] An **abstraction for player i** is a pair $\alpha_i = \langle \alpha_i^{\mathbf{I}}, \alpha_i^A \rangle$, where,*

- *$\alpha_i^{\mathbf{I}}$ is a partitioning of H_i, defining a set of abstract information sets that must be coarser[1] than $\mathbf{I}_i$, and*
- *α_i^A is a function on histories where $\alpha_i^A(h) \subseteq A(h)$ and $\alpha_i^A(h) = \alpha_i^A(h')$ for all histories h and h' in the same abstract information set. We will call this the abstract action set.*

*The **null abstraction** for player i, is $\phi_i = \langle \mathbf{I}_i, A \rangle$. An **abstraction** α is a set of abstractions α_i, one for each player. Finally, for any abstraction α, the **abstract game**, Γ^α, is the extensive game obtained from Γ by replacing $\mathbf{I}_i$ with $\alpha_i^{\mathbf{I}}$ and $A(h)$ with $\alpha_i^A(h)$ when $P(h) = i$, for all i.*

Waugh [Waugh *et al.*, 2009] did an analysis of the effect of abstracting games. In particular, they found that monotonicity in abstraction refinement does not hold. Assume we have two abstractions α_a and α_b of Γ such that α_a is a strict refinement of α_b, in that every information set in α_b is the union of some information sets in α_a. This means that α_a holds strictly more information about the real world than α_b. Waugh found that an equilibrium solution to Γ^{α_a} could be more exploitable in Γ than an equilibrium solution to Γ^{α_b}. In essence, larger abstractions do not necessarily produce better strategies. Our work deals with how to use these abstract solutions to play in the full game and requires a slightly different notion of abstraction.

One problem with using this definition of abstraction is that it requires us to explicitly define how the histories in the real game are partitioned. Sometimes we want to define an abstract game in which we do not explicitly know the partitioning, but rather only that a partitioning with some specific properties exists. Therefore, we define a more general kind of abstraction, called a loose abstraction, for which we defer the definition of the partitioning method.

Definition 3 (Loose Abstraction) *An extensive game Γ' is an **abstraction of** Γ if $H' \subseteq H$ and $\exists$ an abstraction α such that $\forall i$*

- *$\exists$ a bijection between $\mathbf{I}'_i$ and $\alpha_i^{\mathbf{I}}$ and any two histories $h'_1, h'_2 \in H'$ in the same information set in $\mathbf{I}'_i$ are in the same information set in $\alpha_i^{\mathbf{I}}$*
- *$A'(h') = \alpha_i^A(h') \; \forall \; h' \in H'$*

This definition allows us to define an abstract game based solely on restricting a specific set of actions from the real

[1] Recall that partition A is coarser than partition B, if and only if every set in B is a subset of some set in A, or equivalently x and y are in the same set in A if x and y are in the same set in B.

game rather than defining a specific partitioning. The abstract game is defined to contain all histories from the real game except any history containing a restricted action. To use an abstract game strategy to play in the real game we must be able to handle histories in the real game that contain actions that are no longer legal in the abstract game. That is the purpose of translation, which is described in section 3.

2.3 Heads-up No-limit Texas Hold'em

Texas Hold'em poker can be represented as an extensive game and abstraction can be used to reduce the size of the game. Texas Hold'em is a game played with a standard 52 card deck consisting of 4 suits and 13 ranks. The goal of each player is to obtain the best 5-card poker hand according to the standard poker ranking. The game begins by posting the *blinds*: the player to the left of the dealer puts a *small blind* (*e.g.* $1) into the pot and the player two seats to the left of the dealer puts the *big blind* into the pot (*e.g.* $2). Every player is then dealt two cards, followed by a betting round (described later). Three community cards that any player can use are then turned face up in the middle of the table, called the *flop*, followed by another betting round. Two more community cards are dealt face up, called the *turn* and *river*, with betting rounds following each card. Finally, all remaining players reveal their cards and the best five card poker hand wins the pot.

How the betting round works differs slightly depending on the variant being played. In all variants, the pre-flop betting round begins with the person left of the big blind, and all other betting rounds begin with the small blind. In limit hold'em, every player can choose to either *fold* (forfeit their hand), *check*/*call* (match the largest current bet), or *bet*/*raise* (add additional chips to the pot that others must match). The amount raised is determined by the round and not by the players. No-limit differs in that players can bet/raise any number of chips between the minimum bet and all of their chips.

Hold'em poker can be represented by an extensive game, since every card dealt is represented by a chance action and every fold, check/call or bet/raise is represented by a player action. The information set partitions are defined by the fact that each player cannot see the other players' cards, and the utility of each hand is equal to the number of chips won/lost.

In this paper we use a specific variant of two player (heads-up) no-limit Texas Hold'em. This variant has a small blind of $1, a big blind of $2 and stack sizes of $1000. This is the variant used in the no-limit event of the annual computer poker competitions [Zinkevich and Littman, 2006]. Since this variant is quite large, we use abstraction to create a game of manageable size and then solve this abstract game. In the abstraction process we consider both *card abstraction* as well as *action abstraction*. Once we have created the abstract game, we compute an equilibrium using regret minimization [Zinkevich *et al.*, 2008].

The card abstraction is based on bucketing similar hands together. Our bucketing method is *expected hand strength squared* [Johanson, 2007, pg 25-28]. For any given hand, we can roll out all of the remaining cards to find all possible future hands it could become (and the probabilities of those hands occurring). We can then compute the expectation of the square of the final hand strength over all these possible hands, where hand strength refers to the probability that the hand will win the game. We then distribute all of the hands into the n available buckets according to this metric, with the top $1/n\%$ hands going into the first bucket and so on. The specific abstraction we used has 169 buckets on the preflop, 64 on the flop, and 8 on the turn and river. Since there are exactly 169 possible hands one could have on the preflop, taking into account suit isomorphisms, our bucketing on the preflop simply assigns one hand to each bucket. The flop buckets are then created using the expected hand strength squared metric *independent of the preflop buckets*. Our abstraction effectively forgets it preflop bucket once the flop comes. This differs from the turn and river, in which the buckets are calculated dependent upon previous strength buckets. For instance, the 8 turn buckets depend upon the flop buckets, so that a turn bucket actually consists of the pair [flop bucket, raw turn bucket]. We use this card abstraction because it can be used to find a good solution strategy in 24 hours. In fact a strategy that uses this abstraction defeats all of the competitors in the 2007 no-limit competition, if the competition is re-run with it as a participant.

The action abstraction we use works by restricting the number of actions a player can take. The method used by many researchers and first defined by Gilpin [Gilpin *et al.*, 2008] limits every player to 4 actions. Every player can fold (*f*), check/call (*c*), raise pot (*p*), or go all-in (*a*). Raising pot refers to making a bet of the size of the number of chips in the pot, and going all-in refers to betting all of the chips in one's stack. This is an abstraction of the full game in which the actions are restricted to *fcpa*. However, when playing a real game, we must still handle the situation in which our opponent makes, for instance, a bet of 1.5 times the pot. This requires us to translate real states into states in the abstract game.

2.4 Leduc Hold'em

Leduc Hold'em is a game similar to Texas Hold'em but much smaller. The Leduc game only has 6 cards, 2 suits with 3 ranks. Each player is dealt one private card, and the flop consists of only one public card. There are only two betting rounds, one after the private cards are dealt and one after the flop is dealt. The variant we use has stack sizes of 12 chips and has each player ante 1 chip at the start of each hand. Since this game is so small, we can directly calculate the best response to any strategy in this game. This means that given any strategy, we can compute a value that tells us how exploitable that strategy is.

For our experiments using the Leduc game, we use the null card abstraction and allow more betting options in the betting abstraction. In addition to the normal *fcpa* options, we allow a half-pot option (*h*) and a double-pot option (*d*). This makes the largest abstraction *fchpda*.

3 Translation Methods

State translation refers to the process of translating a state in the real game to a state in an abstracted game. In practice an abstraction on chance nodes uses an explicit partition so that

translation is just a table look-up. For example, in poker, it is common to use a hand strength function to partition the two card starting hands into a fixed number of *buckets*, where the hands AA, KK and the other strongest hands are usually in the same bucket. However, the player action space is usually just restricted without explicitly partitioning the real space. In this case, one must convert a real action history into a legal history in the abstract game in order to use the abstract solution. This is most easily done by stepping through the history sequentially and converting every real action into a legal action in the abstract game.

3.1 Hard Translation

The current translation method [Gilpin *et al.*, 2008], which we will refer to as **hard translation**, defines a single translation function that maps a history in the real game to a history in the abstract game.

Definition 4 *A **hard translation function** is a function on histories $T(h) \in H'$ where $h \in H$.*

*A **hard translation in-step function** is a function on histories and actions $t_{in}(h, a) \in A'(T(h))$ where $h \in H, a \in A(h)$.*

*A **hard translation out-step function** is a function on histories and actions $t_{out}(h, a') \in A(h)$ where $h \in H, a' \in A'(T(h))$*

Hard translation provides a partitioning of real-game histories that is sufficient to convert a loose abstraction into an explicit abstraction. A translation function can be used to define the partitioning $\alpha_i^{\mathbf{I}}$ where h, h' are in the same information set iff $T(h) = T(h')$. By explicitly defining T separately from the abstraction, we can vary how a solution to the abstract game plays in the real game without changing the abstraction and thus recomputing the solution. This way, we can evaluate many different translation functions using the same loose abstraction.

A simple way to create a translation function is to step through the action history converting every action to a legal action in the abstract game. This allows us to recursively define the translation function as follows:

$$T((h, a)) = (T(h), t_{in}(h, a)). \tag{1}$$

The step function can be implemented using a similarity metric to define how close an action in the real game is to various actions in the abstract game. If we let $S(h, a, a')$ be a similarity metric where $a \in A(h)$ and $a' \in A'(T(h))$, then we can define the value of the translation step function to be the a' with the highest S value:

$$t_{in}(h, a) = argmax_{a'}(S(h, a, a')) \tag{2}$$

This results in converting every real action in a history to the closest legal action in the abstract game and thus creates a legal abstract history.

After we obtain a history in the abstract game we can sample the solution we have for an action to perform. The purpose of the out-step function is to translate the action in the abstract game into an action in the real game. Although the action the abstract solution provides is usually a legal action in the full game, we may want to perform a slightly different action. This enables our player to take actions in the full game that are not legal actions in the abstract game. However, when doing so we wish to ensure that we maintain internal consistency in our translation. This means that when we translate an earlier action we took in the full game, we always translate it to the abstract action the abstract solution told us to perform. This is guaranteed by forcing the out-function to be the inverse of the in-function:

$$t_{in}(h, t_{out}(h, a')) = a' \tag{3}$$

If we incorrectly translate our previous actions, then it is possible that we could get to game states in which our solution does not know what to do (because it believes it could never get there). Maintaining internal consistency ensures that this will never happen.

The problem with hard translation is that if you know the similarity metric used by your opponent then you can easily exploit their abstraction. For instance, if you know that $t_{in}(h, a) = t_{in}(h, b)$, then you can choose whichever action a or b that benefits you the most, knowing that your opponent will interpret them as the same action. An example of this in poker revolves around translating actions to a pot bet. If I know that my opponent will interpret bets of 1.5*pot and 0.5*pot as pot bets, then I can choose either at will. For instance, when I have a good hand I have a higher probability of winning the pot, and therefore I would want more chips in the pot and would choose 1.5*pot. Similarly, when I have a bad hand I would choose 0.5*pot to risk fewer chips.

The reason why hard translation is so dangerous is that a player does not even need to know the strategy of the opponent to exploit that opponent. The knowledge that the opponent will interpret actions a and b the same is enough. This differs greatly from knowing how the opponent abstracts chance nodes (i.e. cards) since one cannot control the chance nodes. Although one would know that the opponent views two sets of chance nodes as the same, it is difficult to exploit this knowledge without knowing how the opponent plays in that situation.

It is possible that no opponent would understand our translation function enough in order to exploit it. However, it is possible to learn how an agent performs its translation. Assuming an agent is using hard translation, we need only learn which actions it responds to similarly and which it treats differently. We developed a method that can, with high accuracy and within 100 hands, estimate how an agent is performing hard translation. Even if exploiting translation was a more difficult task, the exploitability of a strategy is considered to be one of the best metrics for measuring the strength of a strategy.

3.2 Soft Translation

We propose a new method, which we will refer to as **soft translation**, that takes a history in the real game and returns a weighted set of histories in the abstract game.

Definition 5 *A **soft translation function** is a function on histories $T^p(h) \subseteq \Re \times H'$ where $h \in H$.*

*A **soft translation in-step function** is a function on histories and actions $t^p_{in}(h, a) \subseteq \Re \times A'(T_p(h))$ where $h \in H, a \in A(H)$.*

Again we can step through the action history, except now we convert every real action into a weighted set of abstract actions. By weighting these actions by their (normalized) similarity values, we obtain a more accurate analog of what actually happened in the real game. Since each action in a history is translated to a weighted set of actions, the number of weighted histories grows exponentially as we translate all the real actions in history. This exponential growth can be avoided by sampling the returned action set according to their weights instead of maintaining all of them. In this manner we can view soft translation as a nondeterministic version of hard translation. This non-exponential method is the one we implemented.

We need not define another out-step function, as we can use the previous one after choosing one of the histories returned by soft translation. However, it is slightly more difficult to maintain internal consistency here. First, as the in-function now returns many histories, we modify equation 3 to return a' with weight 1 and all other actions with weight 0. Second, the action we take may only make sense assuming knowledge of the abstract history used by the out-function. This means that in order to avoid confusing ourselves later, we must obtain the same abstract history during translation later in the game. Fortunately, there is a simple solution to this problem. By assigning an ID to every game we play, we can seed our sampling process with a hash of the ID to ensure that, within one game, we will always return the same history given the same input.

The concept of maintaining several histories perhaps makes more sense in situations where the opponent's action is hidden. In attempting to model such a situation, simply assuming the most likely event occurred would result in the player being unprepared when this assumption is incorrect. Instead, one can model the situation by weighting different events according to the probability that they occurred. Our situation differs in that we *know* what our opponent did, but we do not *understand* what that action means. Just as we would describe a motorcycle as a mixture of a bicycle and a car, describing an unknown situation as a mixture of known situations can more accurately describe the real situation.

Unfortunately, maintaining multiple histories gives us no guarantee that our agent will perform the correct action. It is possible that none of the solutions to the returned histories contain the correct response since they simply cannot model the situation accurately enough. Additionally, mixing together the solutions from several histories can be dangerous. The way equilibrium solutions mix their actions is very precise and specific to the game that was solved, and modifying these distributions can result in unpredictable performance. However, when dealing with actions that do not exist in the abstract game to begin with, all guarantees of optimality are lost and we are stuck using methods with unbounded worst case scenarios.

4 Application to Poker

With the translation methods, real game and abstract game defined, all that is needed to implement these methods is a similarity metric and an out-step function. Recalling earlier, the abstraction used in the no-limit game allows each player to fold, call, bet pot, or go all-in (fcpa). This means that every bet must be translated to one of these four actions. The metric used by several of the competitors in the AAAI no-limit poker competitions was described by Gilpin and colleagues [2008] and is formalized here.

Definition 6 *The **geometric similarity** of a real action a and a legal action a' in the abstract game is as follows, where b, b' are the respective bet sizes associated with a, a'.*

$$S(h, a, a') = \begin{cases} b/b' & \text{if } b < b' \\ b'/b & \text{otherwise} \end{cases} \quad (4)$$

This is the metric we use in our translation function.

To define the out-step function, we need to map the legal abstract actions to real bet amounts. We define the pot bet option to be a bet of the size of the current real pot, and the all-in action to be a bet the size of the player's remaining chips in the real game. This distinction needs to be made, because in situations where the real pot size does not match the pot size in the abstract state, a pot bet in the abstract state may be a different size than the real pot size. By using this bet size correlation in the out-step function as well as the similarity metric, we ensure internal consistency.

Knowing the similarity metric we can immediately see how a player using hard translation would interpret certain bets. For instance, if p is the number of chips associated with a pot bet and a is the number of chips associated with an all-in bet, then we know that any bet larger than $\sqrt{p * a}$ will be interpreted as all-in, and any bet smaller than that will be interpreted as a pot bet. Similarly, if we consider a *check* to be a bet of 1, then $\sqrt{p}$ is the border that determines whether a bet is considered a pot bet or a check/call[2]. This means that any amount from $\sqrt{p}$ to $\sqrt{p * a}$ will be interpreted as a pot bet, and we can choose to use whichever one will benefit us the most knowing such a player cannot tell the difference.

A slightly different metric is used for soft translation. Looking at the previous metric, we see that every action will always have a non-zero similarity value. However, since the weights of all actions are important in soft translation, we desire that when the similarity value of one action is 1 that the values of all other actions are 0. Because the different bet sizes lay on the number line, we only need to consider the closest legal bets larger and smaller than the actual bet (all other actions are given weight 0). If $b_1 < b < b_2$ where b is the real bet associated with a and b_1 and b_2 are the bets of the two closest legal abstract actions a_1, a_2, then the metrics are as follows.

$$S(h, a, a_1) = \frac{b_1/b - b_1/b_2}{1 - b_1/b_2} \quad (5)$$

$$S(h, a, a_2) = \frac{b/b_2 - b_1/b_2}{1 - b_1/b_2} \quad (6)$$

Thus, we have that the metric $S(h, a, a_1) = 1$ when $a = a_1$ and $S(h, a, a_1) = 0$ when $a = a_2$, as desired. An important aspect of this property is that if the original history being translated is a legal history in the abstract game, then soft translation will return this history with weight 1.

[2]Since calling affects the game tree differently than a bet, we can only translate real bets into check/calls in certain situations.

5 Results

For our experiments we created two no-limit agents for each variant we used. Within each variant, the two agents use the same solution to the abstracted game, but one uses hard translation and one uses soft translation. In the $1000 stack Texas Hold'em variant, we used the *fcpa* betting abstraction. In the $12 stack Leduc Hold'em game, we used several different betting abstractions. For the different abstractions, we varied whether each pot bet (*hpd*) was allowed or not. This led to 8 abstractions, all of which were used to create players using both soft and hard translation.

Since the Texas Hold'em game is too large to compute a proper best response to our agents, we instead played these agents against a variety of different opponents. Some of the opponents were designed to exploit the normal translation and others simply play using a solution to a different betting abstraction. Note that in this section, a *large* pot bet or a *small* pot bet refers to making the largest or smallest possible bet that our opponent will interpret as a pot bet. This notation will also be used when referencing bet amounts other than a pot bet.

5.1 Opponents

The first set of opponents that were created were designed to exploit the normal translation method. This exploitation is done by controlling the size of the pot in a way that is invisible to a player using the normal translation method. Knowing, for instance, the range of bets that the player interprets as a pot bet allows us to make larger or smaller pot bets and thus control the size of the pot. Controlling the pot size allows us to exploit the player in two ways. First, we can place the player in situations where they will play very poorly. The first opponent, *naïvePA*, works by using this method. Second, we can artificially increase the value of our wins and decrease the cost of our losses by increasing the size of the pot when we are likely to win and decreasing the size of the pot when we are likely to lose. The +- variants use this concept.

The first opponent, *naïvePA*, performs a pure exploitation on the pot and all-in actions without considering the cards it is dealt. This agent check/calls to the flop, after which it will fold to any bet. If its opponent does not bet, it will make a large pot bet. On the turn it will then make a small all-in bet. This methods works because it places the exploited player in a situation it does not understand well and then coerces the player to make a poor decision. By making a large pot bet, naïvePA has the ability to make the pot size drastically larger than the exploited player thinks it is. The player believes that there are far fewer chips in the pot than there actually are, and when faced with an all-in bet it will fold more often than it should. Additionally, naïvePA loses fewer chips than it should, its entire stack, when the player actually calls the all-in bet, making this exploitative strategy even safer.

The next opponent, +-, uses a much more stable exploitation technique. This player uses the same solution as the opponent it faces, except it varies the size of its bets based upon its cards. Specifically, when making a bet it will make a large bet if its hand is in the top 25% of hands and will make a small bet otherwise. This strategy results in the pot being larger when +- has good hands and the pot being smaller when it has poor hands. Similarly, -+ works the same way except it reverses the type of bet it makes based upon its hand. We expect that reversing the +- strategy will have the opposite effect on the amount of money won against the player. Two other opponents, *+1-1* and *-1+1*, are variations on these techniques. When making a bet, these players will instead bet 1 chip more or less depending on the strength of their hand.

Lastly we have two opponents that do not use exploitative techniques. These opponents are simply equilibrium solutions to different betting abstractions. This means that they will take actions that need to be translated by the player, but these actions are not designed to take advantage of how the player's translation method works. *fc75pa* and *fc125pa* bet 75% and 125% of the pot instead of 100% of the pot, respectively. These players do not use the same solution as their opponent, but rather the solution to their own abstracted games.

In summary, *naïvePA*, +- and *+1-1* are all designed to exploit the normal translation method to different degrees. The inverse players, -+ and *-1+1*, are weak agents that manage to hurt themselves by exploiting the translation in the wrong direction. Lastly, *fc75pa* and *fc125pa* are designed to see how well the methods handle bets that are non-exploitative but also not part of the abstraction.

5.2 Data

The results of our experiment in the $1000 stack game are shown in table 1 and the results of the Leduc game are shown in table 2. The $1000 stack players played 10 duplicate[3] matches of 500,000 hands each for a total of 10,000,000 hands. The standard deviation of these matches is shown in the table. The values in the Leduc results are exact computations. It is important to note that in last year's AAAI no-limit competition first place beat second place by 0.22 $/h, and the agent that finished first used hard translation.

	Hard	**Soft**
naïvePA	27.01 $\pm$ 0.07	-5.66 $\pm$ 0.14
+-	5.42 $\pm$ 0.08	1.50 $\pm$ 0.05
-+	-5.16 $\pm$ 0.09	-0.95 $\pm$ 0.08
+1-1	0.15 $\pm$ 0.03	0.11 $\pm$ 0.02
-1+1	-0.14 $\pm$ 0.03	-0.09 $\pm$ 0.01
fc75pa	0.01 $\pm$ 0.03	0.07 $\pm$ 0.04
fc125pa	-0.01 $\pm$ 0.04	-0.02 $\pm$ 0.02

Table 1: Performance results between various $1000 stack players in dollars/hand ($/h)

Looking at the table 1, the $1000 stack players, we see that naïvePA beats the player using hard translation by 27 $/h. This is amazing considering that naïvePA does not look at the cards it is dealt. We also see that naïvePA loses to a player using soft translation by over 5 $/h, a significant amount. Similarly, the +- player beats the hard method by 5.4 $/h. This amount is greatly reduced when played against

[3] A duplicate match refers to playing two matches with the same set of cards, except the players sit in opposite positions in each match (ensuring that they each experience the same situations).

the soft method, down to 1.5 $/h. This shows that soft translation is very effective at defending against these particular exploitative opponents.

Conversely, we see that soft translation does not beat the inverse players by as much as a player using hard translation. This makes sense, since the goal of the new method is to reduce the effect of this type of exploitation. Thus, if it defends against an exploitative method then it likely exploits the inverse method less.

Against the fc75pa player soft translation performed worse, and against the fc125pa player it performed slightly better. However, it appears that this new method does not have the same effect on these players as it does on the exploitative ones. It is likely that since these opponents are playing an equilibrium in their own abstractions, that we cannot completely understand their actions using our abstraction.

	Hard-P1	**Soft-P1**	**Hard-P2**	**Soft-P2**
fchpda	0.41	0.23	0.50	0.41
fcpda	1.18	0.84	1.23	1.12
fchpa	0.76	0.38	0.52	0.43
fchda	0.61	0.25	0.57	0.46
fcha	0.86	0.36	1.06	0.61
fcpa	1.44	0.92	1.04	0.93
fcda	0.84	0.51	1.39	1.13
fca	1.19	0.59	1.61	0.88

Table 2: Exploitability of various $12 stack Leduc Hold'em players in dollars/hand ($/h)

Looking at table 2 we see the best response results for various betting abstractions in Leduc. The abstraction name describes what bets are legal. For instance, *fchda* means that the legal actions are fold, call, half pot, double pot, and all-in. The columns show the exploitability of the agent for each position using both hard and soft translation. Hard-P1 refers to how much a knowledgeable opponent sitting in position 1 could win per hand against a player using the hard translation and the described abstraction. The exploitability for each position is listed to show that soft translation appears to be a strict improvement over hard translation. This is seen by the fact that every value in the Soft-P1 column is smaller than the corresponding value in the Hard-P1 column (and similarly for the P2 columns). This is not to say that there may exist a situation in which it is not a strict improvement, but for the experiments we ran we see that the exploitability of a player using soft translation was always less than the exploitability of a player using hard translation.

6 Conclusion

In this paper we formally described the methods of abstraction and translation used to handle extensive games with large action sets. Additionally, we looked at the current method of translation, described why it could result in an exploitable agent and showed an example of how this can be done in poker. We also described a new probabilistic translation method that helps counter these exploitative techniques. This new method greatly reduced how exploitable the agent was to these techniques. Additionally, this new method was found to produce players that were strictly less exploitable than players produced using the previous method in a small poker game where exploitability can be measured. However, our data also showed that the agent can suffer a performance loss when playing non-exploitative opponents that play using a different action abstraction. It is possible that further development of this technique can reduce or reverse this performance loss.

Acknowledgments

We would like to thank the Computer Poker Research Group at the University of Alberta for their insights and discussions. This research was supported in part by research grants from the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Alberta Informatics Circle of Research Excellence (iCORE).

References

[Billings *et al.*, 2002] Darse Billings, Aaron Davidson, Jonathan Schaeffer, and Duane Szafron. The challenge of poker. *Artificial Intelligence*, 134:2002, 2002.

[Gilpin *et al.*, 2007] Andrew Gilpin, Samid Hoda, Javier Peña, and Tuomas Sandholm. Gradient-based algorithms for finding nash equilibria in extensive form games. In *3rd International Workshop on Internet and Network Economics (WINE)*, 2007.

[Gilpin *et al.*, 2008] Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre Sorensen. A heads-up no-limit texas hold'em poker player: discretized betting models and automatically generated equilibrium-finding programs. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems (AAMAS)*, 2008.

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

[Osborne and Rubenstein, 1994] M. Osborne and A. Rubenstein. *A Course in Game Theory*. The MIT Press, 1994.

[Waugh *et al.*, 2009] Kevin Waugh, David Schnizlein, Michael Bowling, and Duane Szafron. Abstraction pathology in extensive games. In *Proceedings of the 8th international joint conference on Autonomous agents and multiagent systems (AAMAS)*, 2009.

[Zinkevich and Littman, 2006] Martin Zinkevich and Michael Littman. The AAAI computer poker competition. *Journal of the International Computer Games Association*, 29, 2006. News item.

[Zinkevich *et al.*, 2008] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems 20 (NIPS)*, 2008.