Strategy Purification *

Sam Ganzfried, Tuomas Sandholm, and Kevin Waugh
Computer Science Department
Carnegie Mellon University
{sganzfri, sandholm, waugh}@cs.cmu.edu

Abstract

There has been significant recent interest in computing effective practical strategies for playing large games. Most prior work involves computing an approximate equilibrium strategy in a smaller abstract game, then playing this strategy in the full game. In this paper, we present a modification of this approach that works by constructing a *deterministic* strategy in the full game from the solution to the abstract game; we refer to this procedure as *purification*. We show that purification, and its generalization which we call *thresholding*, lead to significantly stronger play than the standard approach in a wide variety of experimental domains. First, we show that purification improves performance in random 4×4 matrix games using random 3×3 abstractions. We observe that whether or not purification helps in this setting depends crucially on the support of the equilibrium in the full game, and we precisely specify the supports for which purification helps. Next we consider a simplifed version of poker called Leduc Hold'em; again we show that purification leads to a significant performance improvement over the standard approach, and furthermore that whenever thresholding improves a strategy, the biggest improvement is often achieved using full purification. Finally, we consider actual strategies that used our algorithms in the 2010 AAAI Computer Poker Competition. One of our programs, which uses purification, won the two-player no-limit Texas Hold'em bankroll division. Furthermore, experiments in two-player limit Texas Hold'em show that these performance gains do not necessarily come at the expense of worst-case exploitability and that our algorithms can actually produce strategies with lower exploitabilities than the standard approach.

1 Introduction

Developing strong strategies for agents in multiagent systems is an important and challenging problem. It has received significant attention in recent years from several different communities, particularly in light of the competitions held at some of the top conferences (e.g., the computer poker, robo-soccer, and trading agent competitions). Most domains of interest are so large that solving them directly (i.e., computing a Nash equilibrium or other relevant solution concept) is computationally infeasible, so some amount of approximation is necessary to produce practical agents.

In particular, significant work has been done in recent years on computing approximate game-theory-based strategies in large games. This work typically follows a three-step approach. First, an *abstraction algorithm* is run on the original game G to construct a smaller game G' which is strategically similar to G (Billings et al. 2003; Gilpin and Sandholm 2007; Shi and Littman 2002). Second, an *equilibrium-finding algorithm* is run on G' to compute an ϵ-equilibrium σ' (Gilpin et al. 2007; Zinkevich et al. 2007). Third, a *reverse mapping* is applied to σ' to compute an approximate equilibrium σ in the full game G (Gilpin, Sandholm, and Sørensen 2008; Schnizlein, Bowling, and Szafron 2009). While most prior work has focused on the first two steps of this approach, in this paper we focus on the third.

Almost all prior work has used the trivial reverse mapping, in which σ is the straightforward projection of σ' into G. In other words, once the abstract game is solved, its solution is just played directly in the full game. However, in some settings this is simply not possible; for instance, if we abstract away some actions of G in G', our strategy must still specify how to react if the opponent selects some of those omitted actions. For example, abstraction algorithms for no-limit Texas Hold'em often involve restricting the set of allowed bet sizes; however when the game is actually played, the opponent is free to make bets of any size. A currently popular way of dealing with this is to apply a randomized reverse mapping that maps the bet size of the opponent to one of the two closest bet sizes in the abstraction (Schnizlein, Bowling, and Szafron 2009).

In this paper, we show that applying more sophisticated reverse mappings can lead to significant performance improvements — even in games where the trivial mapping is possible. The motivation for our approaches is that the exact probabilities of the mixed strategy equilibrium in an abstraction often exemplify overfitting to the particular lossy abstraction used. Ideally, we would like to extrapolate general principles from the strategy rather than just use values that were finely tuned for a specific abstraction. This is akin to the classic example from machine learning, where we would prefer a degree-one polynomial that fits the training data quite well to a degree-hundred polynomial that may

*This material is based upon work supported by the National Science Foundation under grants IIS-0905390, IIS-0964579, and CCF-1101668.

fit it a little better.

We show that our algorithms lead to significantly stronger play in several domains. First, we show that purification improves performance in random 4×4 matrix games using random 3×3 abstractions. We observe that whether or not purification helps in this setting depends crucially on the support of the equilibrium in the full game, and we precisely specify the supports for which purification helps. Next we consider a simplifed version of poker called Leduc Hold'em; again we show that purification leads to a significant performance improvement over the standard approach, and furthermore that whenever thresholding improves a strategy, the biggest improvement is often achieved using full purification. Finally, we consider actual strategies that used our algorithms in the 2010 AAAI Computer Poker Competition. One of our programs, which uses purification, won the two-player no-limit Texas Hold'em bankroll division. Furthermore, experiments in two-player limit Texas Hold'em show that these performance gains do not necessarily come at the expense of worst-case exploitability, and that our algorithms can actually produce strategies with lower exploitabilities than the standard approach.

2 Game theory background

In this section, we briefly review relevant definitions and prior results from game theory and game solving.

2.1 Strategic-form games

The most basic game representation, and the standard representation for simultaneous-move games, is the *strategic form*. A *strategic-form game* (aka matrix game) consists of a finite set of players N, a space of *pure strategies* S_i for each player, and a utility function $u_i : \times S_i \to \mathbb{R}$ for each player. Here $\times S_i$ denotes the space of *strategy profiles* — vectors of pure strategies, one for each player.

The set of *mixed strategies* of player i is the space of probability distributions over his pure strategy space S_i. We will denote this space by Σ_i. Define the *support* of a mixed strategy to be the set of pure strategies played with nonzero probability. If the sum of the payoffs of all players equals zero at every strategy profile, then the game is called *zero sum*. In this paper, we will be primarily concerned with two-player zero-sum games. If the players are following strategy profile σ, we let σ_{-i} denote the strategy taken by player i's opponent, and we let Σ_{-i} denote the opponent's entire mixed strategy space.

2.2 Extensive-form games

An *extensive-form* game is a general model of multiagent decision-making with potentially sequential and simultaneous actions and imperfect information. As with perfect-information games, extensive-form games consist primarily of a game tree; each non-terminal node has an associated player (possibly *chance*) that makes the decision at that node, and each terminal node has associated utilities for the players. Additionally, game states are partitioned into *information sets*, where the player whose turn it is to move cannot distinguish among the states in the same information set. Therefore, in any given information set, a player must choose actions with the same distribution at each state contained in the information set. If no player forgets information that he previously knew, we say that the game has *perfect recall*. A (behavioral) *strategy* for player i, $\sigma_i \in \Sigma_i$, is a function that assigns a probability distribution over all actions at each information set belonging to i.

2.3 Nash equilibria

Player i's *best response* to σ_{-i} is any strategy in $\arg\max_{\sigma'_i \in \Sigma_i} u_i(\sigma'_i, \sigma_{-i})$. A *Nash equilibrium* is a strategy profile σ such that σ_i is a best response to σ_{-i} for all i. An ϵ-*equilibrium* is a strategy profile in which each player achieves a payoff of within ϵ of his best response.

In two player zero-sum games, we have the following result which is known as the *minimax theorem*:

$$v^* = \max_{\sigma_1 \in \Sigma_1} \min_{\sigma_2 \in \Sigma_2} u_1(\sigma_1, \sigma_2) = \min_{\sigma_2 \in \Sigma_2} \max_{\sigma_1 \in \Sigma_1} u_1(\sigma_1, \sigma_2).$$

We refer to v^* as the *value* of the game to player 1. Sometimes we will write v_i as the value of the game to player i. It is worth noting that any equilibrium strategy for a player will guarantee an expected payoff of at least the value of the game to that player.

All finite games have at least one Nash equilibrium. In two-player zero-sum strategic-form games, a Nash equilibrium can be found efficiently by linear programming. In the case of zero-sum extensive-form games with perfect recall, there are efficient techniques for finding an ϵ-equilibrium, such as linear programming (LP), the excessive gap technique (Gilpin et al. 2007), and counterfactual regret minimization (Zinkevich et al. 2007). However, the latter two scale to much larger games: 10^{12} states in the game tree, while the best current LP techniques cannot scale beyond 10^8 states.

2.4 Abstraction

Despite the tremendous progress in equilibrium-finding in recent years, many interesting real-world games are so large that even the best algorithms have no hope of computing an equilibrium directly. The standard approach of dealing with this is to apply an *abstraction* algorithm, which constructs a smaller game that is similar to the original game; then the smaller game is solved, and its solution is mapped to a strategy profile in the original game. The approach has been applied to two-player Texas Hold'em poker, first with a manually generated abstraction (Billings et al. 2003), and now with abstraction algorithms (Gilpin and Sandholm 2007). Many abstraction algorithms work by coarsening the moves of chance, collapsing several information sets of the original game into single information sets of the abstracted game.

The game tree of two-player no-limit Texas Hold'em has about 10^{71} states (while that of two-player limit Texas Hold'em has about 10^{18} states); so significant abstraction is necessary, since currently we can only solve games with up to 10^{12} states.

3 Purification and thresholding

In this section we present our new reverse-mapping algorithms, *purification* and *thresholding*.

Suppose we are playing a game Λ that is too large to solve directly. As described in Section 2.4, the standard approach would be to construct an abstract game Λ', compute an equilibrium σ' of Λ', then play the strategy profile σ induced by σ' in the full game Λ.

One possible problem with doing this is that the specific strategy profile σ' might be very finely tuned for the abstract game Λ', and it could perform arbitrarily poorly in the full game (see the results in Section 5). Ideally we would like to extrapolate the important features from σ' that will generalize to the full game and avoid playing a strategy that is overfit to a particular abstraction. This is the motivation for our new algorithms.

3.1 Purification

Let τ_i be a mixed strategy for player i in a strategic-form game, and let $S = \arg\max_j \tau_i(j)$, where j ranges over all of player i's pure strategies. Then we define the *purification* $\text{pur}(\tau_i)$ of τ_i as follows:

$$\text{pur}(\tau)(j) = \begin{cases} 0 & : \quad j \notin S \\ \frac{1}{|S|} & : \quad j \in S \end{cases}$$

Informally, this says that if τ_i plays a single pure strategy with highest probability, then the purification will play that strategy with probability 1. If there is a tie between several pure strategies of the maximum probability played under τ_i, then the purification will randomize equally between all maximal such strategies. Thus the purification will usually be a pure strategy, and will only be a mixed strategy in degenerate special cases when several pure strategies are played with identical probabilities.

If τ_i is a behavioral strategy in an extensive-form game, we define the purification similarly: at each information set I, $\text{pur}(\tau_i)$ will play the purification of τ_i at I.

3.2 Thresholding

Purification can sometimes seem quite extreme: for example, if τ_i plays action a with probability 0.51 and action b with probability 0.49, τ will still never play b. Maybe we would like to be a bit more conservative, and only set a probability to 0 if it is below some threshold ϵ. We refer to this algorithm as *thresholding*.

More specifically, thresholding will set all actions that have weight below ϵ to 0, then renormalize the remaining action probabilities. One intuitive interpretation of thresholding is that actions with probability below ϵ were just given positive probability due to noise from the abstraction (or because an anytime equilibrium-finding algorithm had not yet taken those probabilities all the way to zero), and really should not be played in the full game.

4 Evaluation metrics

In recent years, several different metrics have been used to evaluate strategies in large games.

4.1 Empirical performance

The first metric, which is perhaps the most meaningful, is *empirical performance* against other realistic strategies. For example, in the annual computer poker competition at AAAI, programs submitted from researchers and hobbyists from all over the world compete against one another. Empirical performance is the metric we will be using in Section 8 when we assess our performance in Texas Hold'em.

4.2 Worst-case exploitability

The worst-case *exploitability* of player i's strategy σ_i is the difference between the value of the game to player i and the payoff when the opponent plays his best response to σ_i (aka his *nemesis strategy*). Formally it is defined as follows:

$$\text{expl}(\sigma_i) = v_i - \min_{\sigma_{-i} \in \Sigma_{-i}} u_i(\sigma_i, \sigma_{-i}).$$

Worst-case exploitability has recently been used to assess strategies in a simplified variants of poker (Gilpin and Sandholm 2008; Waugh et al. 2009).

Any equilibrium has zero exploitability, since it receives payoff v_i against its nemesis. So if our goal were to approximate an equilibrium of the full game, worst-case exploitability would be a good metric to use, since it approaches zero as the strategy approaches equilibrium.

Unfortunately, the worst-case exploitability metric has several drawbacks. First, it cannot be computed in very large games. For example, it cannot currently be computed in two-player no-limit Texas Hold'em.

Second, exploitability is a worst-case metric that implicitly assumes that the opponent is both trying to exploit our strategy and that he is able to do that effectively in the full game. In many large games, agents just play fixed strategies since the number of interactions is generally tiny compared to the size of the game, and it is usually quite difficult to learn to effectively exploit opponents online. For example, in recent computer poker competitions, almost all submitted programs simply play a fixed strategy. In the 2010 AAAI computer poker competition, many of the entrants attached summaries describing their algorithm. Of the 17 bots for which summaries were included, 15 played fixed strategies, while only 2 included some element of attempted exploitation. If the opponents are just playing a fixed strategy and not trying to play a best response, then worst-case exploitability is too pessimistic of an evaluation metric. Furthermore, if the opponents all have computational limitations and use abstractions, then they will not be able to fully exploit us in the full game.

4.3 Performance against full equilibrium

In this paper, we will also evaluate strategies based on performance against equilibrium in the full game. The intuition behind this is that in many large two-player zero-sum games, the opponents are simply playing fixed strategies that attempt to approximate an equilibrium of the full game (using some abstraction). For example, most entrants in the annual computer poker competition do this. Against such static opponents, worst-case exploitability is not very significant, as the agents are not generally adapting to exploit us.

This metric, like worst-case exploitability, is not feasible to apply on large games like Texas Hold'em. However, we can still apply it to smaller games as a means of comparing different solution techniques. In particular, we will use this metric in Sections 6 and 7 when presenting our experimental results on random matrix games and Leduc Hold'em. This metric has similarly been used on solvable problem sizes in the past to compare abstraction algorithms (Gilpin and Sandholm 2008).

5 Worst-case analysis

So which approach is best: purification, thresholding, or the standard abstraction approach? It turns out that using the performance against full equilibrium metric, there exist games for which each technique can outperform each other. Thus, from a worst-case perspective, not much can be said in terms of comparing the approaches.

Proposition 1 shows that, for *any* equilibrium-finding algorithm, there exists a game and an abstraction such that purification does arbitrarily better than the standard approach.

Proposition 1. *For any equilibrium-finding algorithms A and A', and for any $k > 0$, there exists a game Λ and an abstraction Λ' of Λ, such that*

$$u_1(pur(\sigma'_1), \sigma_2) \geq u_1(\sigma'_1, \sigma_2) + k,$$

where σ' is the equilibrium of Λ' computed by algorithm A', and σ is the equilibrium of Λ computed by A.

	L	M	R
U	2	0	$-3k-1$
D	0	1	-1

Figure 1: Two-player zero-sum game used in the proof of Proposition 1.

Proof. Consider the game in Figure 1. Let Λ denote the full game, and let Λ' denote the abstraction in which player 2 (the column player) is restricted to only playing L or M, but the row player's strategy space remains the same. Then Λ' has a unique equilibrium in which player 1 plays U with probability $\frac{1}{3}$, and player 2 plays L with probability $\frac{1}{3}$. Since this is the unique equilibrium, it must be the one output by algorithm A'. Note that player 1's purification $\text{pur}(\sigma'_1)$ of σ' is the pure strategy D.

Note that in the full game Λ, the unique equilibrium is (D,R), which we denote by σ. As before, since this equilibrium is unique it must be the one output by algorithm A. Then we have

$$\begin{aligned} u_1(\sigma'_1, \sigma_2) = \frac{1}{3}(-3k-1) + \frac{2}{3}(-1) &= -k-1 \\ u_1(\text{pur}(\sigma'_1), \sigma_2) &= -1. \end{aligned}$$

So $u_1(\sigma'_1, \sigma_2) + k = -1$, and therefore

$$u_1(\text{pur}(\sigma'_1), \sigma_2) = u_1(\sigma'_1, \sigma_2) + k.$$

□

Due to limited space, we omit our other results, but we can similarly show that purification can also do arbitrarily worse against the full equilibrium than standard abstraction, and that both procedures can do arbitrarily better or worse than thresholding (using any threshold cutoff).

6 Random matrix games

The first set of experiments we conducted to demonstrate the power of purification was on random matrix games. This is perhaps the most fundamental and easy to analyze class of games, and is a natural starting point when analyzing new algorithms.

6.1 Evaluation methodology

We studied random 4×4 two-player zero-sum matrix games with payoffs drawn uniformly at random from [-1,1]. We repeatedly generated random games and analyzed them using the following procedure. First, we computed an equilibrium of the full 4×4 game Λ; denote this strategy profile by σ^F. Next, we constructed an abstraction Λ' of Λ by ignoring the final row and column of Λ. As in Λ, we computed an equilibrium σ^A of Λ'. We then compared $u_1(\sigma_1^A, \sigma_2^F)$ to $u_1(\text{pur}(\sigma_1^A), \sigma_2^F)$.

Unfortunately we realized that obtaining statistically significant results could require millions of trials even on small games. In particular, the standard algorithm for solving two-player zero-sum games involves solving a linear program (Dantzig 1951), and solving millions of linear programs would be very time-consuming. Thus, we develop our own algorithm for solving small matrix games that avoids needing to solve linear programs. Our algorithm is similar to the support enumeration of (Porter, Nudelman, and Shoham 2008), but it uses analytical solutions instead of solving linear feasibility programs and therefore runs much faster. Full details of our algorithm are given in the appendix.

6.2 Results

In our experiments on 4×4 random games, we performed 3 million trials, of which 867,110 did not satisfy the conditions of Proposition 4 and thus counted towards our results. The results are given in Table 1. We conclude that purified abstraction outperforms the standard unpurified abstraction approach using 95% confidence intervals. Note that the payoffs listed in the table are not unbiased estimators of actual payoffs of the two approaches over all random games; recall that we ignored certain games for which the two approaches perform identically in order to reduce the number of trials required. Thus, these payoffs should not be interpreted in terms of their absolute values, but rather should be viewed relatively to one another.

$u_1(\sigma_1^A, \sigma_2^F)$	$u_1(\text{pur}(\sigma_1^A), \sigma_2^F)$
-0.16284 ± 0.00052	-0.14883 ± 0.00061

Table 1: Results for experiments on random 4×4 matrix games. The $\pm$ given is the 95% confidence interval.

To understand these results further, we investigated whether they would vary for different supports of σ_F. In particular, we ran Algorithm 3, keeping separate tallies of the performance of pur(σ_1^A) and σ_1^A for each support of σ^F. We observed that pur(σ_1^A) outperformed σ_1^A on many of the supports, while they performed equally on some (and σ_1^A did not outperform pur(σ_1^A) on any, using 95% confidence intervals). A summary of the results from these experiments is given in Observation 1.

Observation 1. *In random 4×4 matrix games using 3×3 abstractions, pur(σ_1^A) performs better than σ_1^A using a 95% confidence interval for each support of σ_F except for supports satisfing one of the following conditions, in which case neither pur(σ_1^A) nor σ_1^A performs significantly better:*

- *σ_F is the pure strategy profile in which each player plays his fourth pure strategy*
- *σ_F is a mixed strategy profile in which player 1's support contains his fourth pure strategy, and player 2's support does not contain his fourth pure strategy.*

We find it very interesting that there is such a clear pattern in the support structures for which pur(σ_1^A) outperforms σ_1^A. We obtained identical results using 3×3 games with 2×2 abstractions, though we did not experiment on games larger than 4×4. We conjecture that similar results would hold on larger games as well and present the general case as an open problem.

7 Leduc Hold'em

Leduc Hold'em is a small poker game that has been used in previous work to evaluate imperfect information game playing techniques (Waugh et al. 2009). Leduc Hold'em is large enough that abstraction has a non-trivial impact, but unlike larger games of interest, e.g., Texas Hold'em, it is small enough that equilibrium solutions in the full game can be quickly computed. That is, Leduc Hold'em allows for rapid and thorough evaluation of game playing techniques against a variety of opponents, including an equilibrium opponent or a best responder.

Prior to play, a deck of six cards containing two Jacks, two Queens, and two Kings is shuffled and each player is dealt a single private card. After a round of betting, a public card is dealt face up for both players to see. If either player pairs this card, he wins at showdown; otherwise the player with the higher ranked card wins. For a complete description of the betting, we refer the reader to (Waugh et al. 2009).

7.1 Experimental evaluation and setup

To evaluate the effects of purification and thresholding in Leduc Hold'em, we compared the performance of a number of abstract equilibrium strategies altered to varying degrees by thresholding against a single equilibrium opponent averaged over both positions. The performance of a strategy (denoted EV for expected value) was measured in millibets per hand (mb/h), where one thousand millibets is a small bet. As the equilibrium opponent is optimal, the best obtainable performance is 0 mb/h. Note that the expected value computations in this section are exact.

We used card abstractions mimicking those produced by state-of-the-art abstraction techniques to create our abstract equilibrium strategies. Specifically, we used the five Leduc Hold'em card abstractions from (Waugh et al. 2009), denoted *JQK*, *JQ.K*, *J.QK*, *J.Q.K* and *full*. The abstraction *full* denotes the null abstraction (i.e., the full unabstracted game). The names of the remaining abstractions consist of groups of cards separated by periods. All cards within a group are indistinguishable to the player prior to the flop. For example, when a player using the *JQ.K* abstraction is dealt a card, he will know only if that card is a king, or if it is not a king. These abstractions can only distinguish pairs on the flop. By pairing these five card abstractions, one abstraction per player, we learned twenty four abstract equilibrium strategies using linear programming techniques. For example, the strategy *J.Q.K-JQ.K* denotes the strategy where our player of interest uses the *J.Q.K* abstraction and he assumes his opponent uses the *JQ.K* abstraction.

7.2 Purification vs. no purification

In Table 2 we present the performance of the regular and purified abstract equilibrium strategies against the equilibrium opponent. We notice that purification improves the performance in all but 5 cases. In many cases this improvement is quite substantial. In the cases where it does not help, we notice that at least one of the players is using the *JQK* card abstraction, the worst abstraction in our selection. Prior to purification, the best abstract equilibrium strategy loses at 43.8 mb/h to the equilibrium opponent. After purification, 14 of the 24 strategies perform better than the best unpurified strategy, the best of which loses at only 1.86 mb/h. That is, only five of the strategies that were improved by purification failed to surpass the best unpurified strategy.

7.3 Purification vs. thresholding

In Figure 2 we present the results of three abstract equilibrium strategies thresholded to varying degrees against the equilibrium opponent. We notice that, the higher the threshold used the better the performance tends to be. Though this trend is not monotonic, all the strategies that were improved by purification obtained their maximum performance when completely purified. Most strategies tended to improve gradually as the threshold was increased, but this was not the case for all strategies. As seen in the figure, the *JQ.K-JQ.K* strategy spikes in performance between the thresholds of 0.1 and 0.15.

From these experiments, we conclude that purification tends to improve the performance of an abstract equilibrium strategy against an unadaptive equilibrium opponent in Leduc Hold'em. Though thresholding is itself helpful, it appears that whenever thresholding improves a strategy, the improvement generally increases monotonically with the threshold, with the biggest improvement achieved using purification.

8 Texas Hold'em

In the 2010 AAAI computer poker competition, the CMU team (Ganzfried, Gilpin, and Sandholm) submitted bots that

Strategy	Base EV	Purified EV	Improvement
JQ.K-J.QK	-119.46	-37.75	81.71
J.QK-full	-115.63	-41.83	73.80
J.QK-J.Q.K	-96.66	-27.35	69.31
JQ.K-J.Q.K	-96.48	-28.76	67.71
JQ.K-full	-99.30	-39.13	60.17
JQ.K-JQK	-80.14	-24.50	55.65
JQ.K-JQ.K	-59.97	-8.31	51.66
J.Q.K-J.QK	-60.28	-13.97	46.31
J.Q.K-J.Q.K	-46.23	-1.86	44.37
J.Q.K-JQ.K	-44.61	-3.85	40.76
full-JQK	-43.80	-10.95	32.85
J.QK-J.QK	-96.60	-67.42	29.18
J.QK-JQK	-95.69	-67.14	28.55
full-J.QK	-52.94	-24.55	28.39
J.QK-JQ.K	-77.86	-52.62	25.23
J.Q.K-full	-68.10	-46.43	21.66
full-JQ.K	-55.52	-36.38	19.14
full-J.Q.K	-51.14	-40.32	10.82
JQK-J.QK	-282.94	-279.44	3.50
JQK-full	-273.87	-279.99	-6.12
JQK-J.Q.K	-258.29	-279.99	-21.70
J.Q.K-JQK	-156.35	-188.00	-31.65
JQK-JQK	-386.89	-433.64	-46.75
JQK-JQ.K	-274.69	-322.41	-47.72

Table 2: Effects of purification on performance of abstract strategies against an equilibrium opponent in mb/h.

used both purification and thresholding to the two-player no-limit Texas Hold'em division. We present the results in Section 8.1. Next, in Section 8.2, we observe how varying the amount of thresholding used effects the exploitabilities of two bots submitted to the 2010 two-player limit Texas Hold'em division.

8.1 Performance in practice

The two-player no-limit competition consisted of two sub-competitions with different scoring rules. In the *instant-runoff* scoring rule, each pair of entrants plays against each other, and the bot with the worst head-to-head record is eliminated. This procedure is continued until only a single bot remains. The other scoring rule is known as *total bankroll*. In this competition, all entrants play against each other and are ranked in order of their total profits. While both scoring metrics serve important purposes, the total bankroll competition is considered by many to be more realistic, as in many real-world multiagent settings the goal of agents is to maximize total payoffs against a variety of opponents.

We submitted bots to both competitions: Tartanian4-IRO (IRO) to the instant-runoff competition and Tartanian4-TBR (TBR) to the total bankroll competition. Both bots use the same abstraction and equilibrium-finding algorithms. They differ only in their reverse-mapping algorithms: IRO uses thresholding with a threshold of 0.15 while TBR uses purification. IRO finished third in the instant-runoff competition, while TBR finished first in the total bankroll competition.

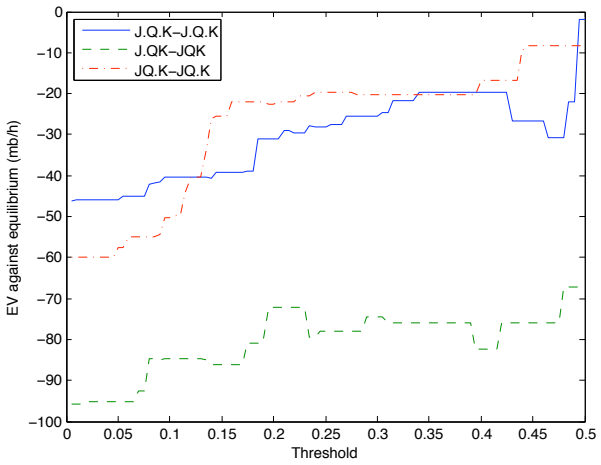

Figure 2: Effects of thresholding on performance of abstract strategies against an equilibrium opponent in mb/h.

Although the bots were scored only with respect to the specific scoring rule and bots submitted to that scoring rule, all bots were actually played against each other, enabling us to compare the performances of TBR and IRO. Table 3 shows the performances of TBR and IRO against all of the bots submitted to either metric in the 2010 two-player no-limit Texas Hold'em competition.

One obvious observation is that TBR actually beat IRO when they played head-to-head (at a rate of 80 milli big blinds per hand). Furthermore, TBR performed better than IRO against every single opponent except for one (c4tw.iro). Even in the few matches that the bots lost, TBR lost at a lower rate than IRO. Thus, even though TBR uses less randomization and is perhaps more exploitable in the full game, the opponents submitted to the competition were either not trying or not able to find successful exploitations. Additionally, TBR would have still won the total bankroll competition even if IRO were also submitted.

These results show that purification can in fact yield a big gain over thresholding (with a lower threshold) even against a wide variety of realistic opponents in very large games.

8.2 Worst-case exploitability

Despite the performance gains we have seen from purification and thresholding, it is possible that these gains come at the expense of worst-case exploitability (see Section 4.2). Exploitabilities for several variants of a bot we submitted to the 2010 two-player limit AAAI computer poker competition (GS6.iro) are given in Table 4. Interestingly, using no rounding at all produced the most exploitable bot, while the least exploitable bot used a threshold of 0.15. Hyperborean.iro was submitted by the University of Alberta to the competition; exploitabilities of its variants are shown as well. Interestingly, Hyperborean's exploitabilities increased monotonically with threshold, with no rounding producing the least exploitable bot.

	c4tw.iro	c4tw.tbr	Hyperborean.iro	Hyperborean.tbr	PokerBotSLO	SartreNL	IRO	TBR
IRO	5334 ± 109	8431 ± 156	-248 ± 49	-364 ± 42	108 ± 46	-42 ± 38		-80 ± 23
TBR	4754 ± 107	8669 ± 168	-122 ± 38	-220 ± 39	159 ± 40	13 ± 33	80 ± 23	

Table 3: Results from the 2010 AAAI computer poker competition for 2-player no limit Texas Hold'em. Values are in milli big blinds per hand (from the row player's perspective) with 95% confidence intervals shown. IRO and TBR both use the same abstraction and equilibrium-finding algorithms. The only difference is that IRO uses thresholding with a threshold of 0.15 while TBR uses purification.

These results show, on the one hand, that it can be hard to predict the relationship between the amount of rounding and the worst-case exploitability, and that it may depend heavily on the abstraction and/or equilibrium-finding algorithm used. While exploitabilities for Hyperborean are perhaps in line with what we might intuitively expect, results from GS6 show that the minimum exploitability can actually be produced by an intermediate threshold value. The reason is that (1) a bot that uses too high a threshold may not have enough randomization and thus be too predictable and reveal too much about its private signals (cards) via its actions, but (2) a bot that uses too low of a threshold may have a strategy that is overfit to the particular abstraction used.

Threshold	**Exploitability of GS6**	**Exploitability of Hyperborean**
None	463.591	**235.209**
0.05	326.119	243.705
0.15	**318.465**	258.53
0.25	335.048	277.841
Purified	349.873	437.242

Table 4: Results for full-game worst-case exploitabilities of several strategies in two-player limit Texas Hold'em. Results are in milli big blinds per hand. Bolded values indicate the lowest exploitability achieved for each strategy.

9 Conclusions and future research

We presented two new reverse-mapping algorithms for large games: purification and thresholding. From a theoretical perspective, we proved that it is possible for each of these algorithms to help (or hurt) arbitrarily over the standard abstraction approach, and each can perform arbitrarily better than the other. However, in practice both purification and thresholding seem to consistently help over a wide variety of domains, with purification generally outperforming thresholding.

Our experiments on random matrix games show that, perhaps surprisingly, purification helps even when random abstractions are used. Our experiments on Leduc Hold'em show that purification leads to improvements on most abstractions, especially as the abstractions become more sophisticated. Additionally, we saw that thresholding generally helps as well, and its performance improves overall as the threshold cutoff increases, with optimal performance usually achieved at full purification. We also saw that purification outperformed thresholding with a lower threshold cutoff in the AAAI computer poker competition against a wide variety of realistic opponents. In particular, our bot that won the 2010 two-player no-limit Texas Hold'em bankroll competition used purification. Finally, we saw that these performance gains do not necessarily come at the expense of worst-case exploitibility, and that intermediate threshold values can actually produce the lowest exploitability.

References

Billings, D.; Burch, N.; Davidson, A.; Holte, R.; Schaeffer, J.; Schauenberg, T.; and Szafron, D. 2003. Approximating game-theoretic optimal strategies for full-scale poker. In *IJCAI*.

Dantzig, G. 1951. A proof of the equivalence of the programming problem and the game problem. In Koopmans, T., ed., *Activity Analysis of Production and Allocation*.

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

Gilpin, A., and Sandholm, T. 2008. Expectation-based versus potential-aware automated abstraction in imperfect information games: An experimental comparison using poker. In *AAAI*. Short paper.

Gilpin, A.; Hoda, S.; Peña, J.; and Sandholm, T. 2007. Gradient-based algorithms for finding Nash equilibria in extensive form games. In *WINE*.

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

Porter, R.; Nudelman, E.; and Shoham, Y. 2008. Simple search methods for finding a Nash equilibrium. *Games and Economic Behavior*.

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

Shi, J., and Littman, M. 2002. Abstraction methods for game theoretic poker. In *Revised Papers from the Second International Conference on Computers and Games*.

von Stengel, B. 2002. Computing equilibria for two-person games. In Aumann, R., and Hart, S., eds., *Handbook of game theory*.

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

Zinkevich, M.; Bowling, M.; Johanson, M.; and Piccione,

C. 2007. Regret minimization in games with incomplete information. In *NIPS*.

Appendix

In this appendix we describe the algorithm used to solve 4×4 two-player zero-sum matrix games for the experiments in Section 6.1. First, we recall from prior work (von Stengel 2002) that in our random matrix game setting, a game will have an equilibrium with *balanced supports* (i.e., equal support sizes for both players) with probability 1.

Definition 1. *A two-player strategic-form game is called* nondegenerate *if the number of pure best responses to a mixed strategy never exceeds the size of its support.*

Definition 2. *A strategic-form game is called* generic *if each payoff is drawn randomly and independently from a continuous distribution.*

Proposition 2. *A generic two-player strategic-form game is nondegenerate with probability 1.*

Corollary 1. *A generic two-player strategic-form game contains a Nash equilibrium with equal support sizes for all players with probability 1.*

Corollary 1 allows us to restrict our attention to balanced supports. For support size at most 3, it turns out that there is a simple closed-form solution for any equilibrium.

Proposition 3. *Let Λ be a nondegenerate two-player strategic-form game. Let S_1 and S_2 be sets of pure strategies of players 1 and 2 such that $|S_1| = |S_2| \leq 3$. Then if Λ contains a Nash equilibrium with supports S_1 and S_2, then there is a simple closed-form solution for the equilibrium.*

Finally, before presenting our algorithm, we note that purification and abstraction will perform identically in games with equilibria that have certain support structures. If we include all of these games, then we will require more samples to differentiate the performances of the two algorithms. On the other hand, if we ignore games for which the two approaches perform identically, then we can differentiate their performances to a given level of statistical significance using fewer samples, and therefore reduce the overall running time of our algorithm. Proposition 4 gives us a set of conditions under which we can omit games from consideration.

Proposition 4. *Let Λ be a two-player zero-sum game, and let Λ' be an abstraction of Λ. Let σ^F and σ^A be equilibria of Λ and Λ' respectively. Then $u_1(\sigma_1^A, \sigma_2^F) = u_1(pur(\sigma_1^A), \sigma_2^F)$ if either of the following conditions is met:*

1. *σ_A is a pure strategy profile*
2. *$support(\sigma_1^A) \subseteq support(\sigma_1^F)$*

Proof. If the first condition is met, then $\text{pur}(\sigma_1^A) = \sigma_1^A$ and we are done. Now suppose the second condition is true and let $s, t \in support(\sigma_1^A)$ be arbitrary. This implies that $s, t \in support(\sigma_1^F)$ as well, which means that $u_1(s, \sigma_2^F) = u_1(t, \sigma_2^F)$, since a player is indifferent between all pure strategies in his support at an equilibrium. Since s and t were arbitrary, player 1 is also indifferent between all strategies in $support(\sigma_1^A)$ when player 2 plays σ_2^F. Since purification will just select one strategy in $support(\sigma_1^A)$, we are done. □

We are now ready to present our algorithm; it is similar to the support enumeration algorithm of (Porter, Nudelman, and Shoham 2008), though it avoids solving linear feasibility programs and omits the conditional dominance tests. The procedure *Test-Feasibility* tests whether an equilibrium exists with the specified support. *Compute-Equilibrium* iterates over all balanced supports and tests whether there is an equilibrium consistent with each one. Finally, our main algorithm repeatedly generates random games and compares the payoffs of σ_1^A and $\text{pur}(\sigma_1^A)$ against the full equilibrium strategy of player 2. As discussed above, to reduce the number of samples needed we omit games for which the equilibria satisfy either condition of Proposition 4. Note that *Compute-Equilibrium* does not actually compute an equilibrium if the only equilibrium is fully mixed for each player; instead it returns a dummy equilibrium profile where each player puts weight 0.25 on each action. We do this because games with only fully mixed equilibria will satisfy the second condition of Proposition 4 and will be ignored by Algorithm 3 anyway.

Algorithm 1 Test-Feasibility(A, S_1, S_2)

```
σ ← candidate solution for supports S_1, S_2 given by closed form expression described in Proposition 3.
if all components of σ are in [0,1] and neither player can profitably deviate then
    return σ
else
    return INFEASIBLE
end if
```

Algorithm 2 Compute-Equilibrium(A)

```
dummy-equilibrium ← ((0.25, 0.25, 0.25, 0.25),(0.25, 0.25, 0.25, 0.25))
for all balanced support profiles S_1, S_2 in increasing order of size, starting with (1,1), (1,2), ... do
    if both supports have size 4 then
        return dummy-equilibrium
    end if
    σ ← Test-Feasibility(A, S_1, S_2)
    if σ ≠ INFEASIBLE then
        return σ
    end if
end for
```

Algorithm 3 Simulate(T)

```
π^A = 0, π^P = 0
for i = 1 to T do
    Λ ← random 4 × 4 matrix game with payoffs in [-1,1]
    Λ' ← 3×3 abstraction of Λ ignoring final pure strategy of each player
    σ^F ← Compute-Equilibrium(Λ)
    σ^A ← Compute-Equilibrium(Λ')
    if σ^F, σ^A do not satisfy either condition of Proposition 4 then
        π^A ← π^A + u_1(σ_1^A, σ_2^F)
        π^P ← π^P + u_1(pur(σ_1^A), σ_2^F)
    end if
end for
```