Using Fictitious Play to Find Pseudo-Optimal Solutions for Full-Scale Poker

William Dudziak

Department of Computer Science, University of Akron Akron, Ohio 44325-4003

Abstract

A pseudo-optimal solution to the poker variant, Two-Player Limit Texas Hold'em was developed and tested against existing world-class poker algorithms. Techniques used in creating the pseudo-optimal solution were able to simplify the problem from complexity from $O(10^{18})$ to $O(10^{7})$. To achieve this reduction, bucketing/grouping techniques were employed, as were methods replacing the chance nodes in the game tree; reducing it from a tree with millions of billions of terminal nodes, to a game tree with only a few thousand.

When played in competition against several world-class algorithms, our algorithm displayed strong results, gaining and maintaining leads against each of the opponents it faced. Using proper abstraction techniques it is shown that we are able to succeed in approaching Nash Equilibria in complex game theoretical problems such as full-scale poker.

Keywords

Fictitious play, Poker, Nash equilibrium, Game theory, Game tree reduction.

1 Introduction

Poker is a competitive card game formed of imperfect decisions punctuated by moments of chance. The variation of poker described in this article is known as *Texas Hold'em*. The game begins when each player is dealt 2 cards face-down. Following the deal, a round of betting proceeds where players have 3

options: call the current bet, and continue the game; bet, and raise the stakes of the game; or fold and remove themselves from the game at the benefit of not having to pay anything further. This first round of betting is referred to as Preflop. After the preflop betting round, three cards are placed face up on the board; all cards placed on the board are shared by all the players. A round of betting continues, referred to as The Flop. Following the flop, a fourth card is placed (The Turn), followed by a round of betting, and lastly a fifth card is place on the table (The River) followed by a final round of betting. If more than one opponent is left at this stage in the game, a showdown occurs where the remaining players display their cards, and must form the most valuable 5 card hand using their 2 hidden cards, and the 5 community cards. The player with the most valuable hand wins all the chips placed in the pot throughout the game.

To Summarize:

- 1. Deal (2 cards each)
- 2. Betting (preflop domain)
- 3. Deal (3 cards shared between all players)
- 4. Betting (flop domain)
- 5. Deal (1 card shared between all players)
- 6. Betting (turn domain)
- 7. Deal (1 card shared between all players)
- 8. Betting (river domain)

Poker in general is of great interest to computer research and artificial intelligence since, in a very well-defined form, it exhibits many of the properties of real world situations including forced action under uncertainty of the current-world state, and being forced to act when the future state is uncertain. The methods for solving poker can provide insight into developing applicable strategies for the development of more versatile artificial intelligence algorithms. The attempt to achieve near-optimal solutions to the game of poker is not a new concept; however the approach presented in this paper is unique, and contains several qualities which are distinct from previously developed pseudo-optimal solutions. Notably, in the domain of Texas Hold'em, an algorithm was developed in 2003 by a research group at the *University of Alberta* named *PsOpti*. *PsOpti* has been shown to have extraordinarily strong performance versus human opponents and has bested all previously developed algorithms [2]. In section 6, results of competition between our algorithm, which we refer to as *Adam* and *PsOpti* are presented.

2. Game Theory

2.1 Definition of Game-Theoretic Optimality

Game theory is an economic study of the strategic interactions between intelligent agents. Through analysis of this interaction, proper behaviors can be derived to yield behavioral strategies that minimize an agent's losses. Such game-theoretic solutions are referred to as optimal, or as a Nash Equilibrium of the given game.

A game-theoretic optimal solution is a fixed behavioral strategy containing a mix of random probabilities at decision nodes within the game tree. Following the stated strategy in an evenly balanced game will yield at worst a break-even situation over long-term play. However, if the opponent is playing sub-optimally, and continues to make strategically dominated errors, an optimal player will be able to exploit these. If an opponent's errors persist in the long-term, such a player will have no chance of winning against an optimal strategy.

The key disadvantage of playing an optimal strategy is that the optimal play only accrues an advantage over opponents only when opponents make *dominated errors*. For example: if there are three choices facing a human player at a given decision node, and the optimal strategy states to play choice A 20% of the time, choice B 0% of the time, and choice C 80% of the time; if the human player were to choose action B, then that would be a dominated mistake, and the optimal opponent would gain an advantage. However, if the opponent were to choose A or C at any frequency (0%-100% of the time), this is a strategic error known as a *non-dominated error*; though this strategy may be suboptimal, an optimal

player will be unable to gain any advantage from this behavior.

An Example to Illustrate the Properties of Nash Equilibria and *Dominated/Non-Dominated Error*:

The game Rock-Paper-Scissors (paper beats rock, rock beats scissors, scissors beats paper), has a remarkably simple optimal solution: play rock with 1/3 probability, paper with 1/3 probability, and scissors with 1/3 probability.

Using Rock-Paper-Scissors, it should be apparent that a strategy of 'always play rock', is not a preferred solution. However, if playing against an optimal opponent, the strategy will not incur any penalties since the player will continue to win 1/3 of the time, lose 1/3 and tie 1/3. This is an illustration of *non-dominated error*. The player is not playing by the rules of the optimal solution, however since the optimal solution involves non-zero probabilities of playing rock/paper/scissors, any strategy involving those elements will not sustain any penalty when playing against an optimal opponent.

A fourth element can be added to this game to demonstrate *dominated error*. We can call the game Rock-Paper-Scissors-Dynamite (the only change to the rules is that dynamite beats rock, and is beaten by paper or scissors). The optimal strategy given these rules is: play rock with 1/3 probability, paper with 1/3 probability, scissors with 1/3 probability, and dynamite with 0 probability. If playing against an optimal opponent and the decision is made to play dynamite, this incurs a dominated error, and the projected winnings from the game will decrease as result.

After this example, it seems that playing dominated errors should be a rare occurrence in games, since the decision seems so clear cut. However, testing has shown that in complicated games, especially games of imperfect information, dominated errors occur often enough (even among pseudo-optimal players), that if played over the longterm, weaknesses in strategy are evident [2].

2.2 Optimality through Fictitious Play

2.2.1 Definition of Fictitious Play

Fictitious play is a set of learning rules designed to produce agents capable of approaching optimality, and was first introduced by G.W. Brown in 1951 [1]. The basic rules of fictitious play are: 1. that each player analyzes their opponent's strategy, and devises a best response. 2. Once a best response is calculated, it is incorporated into or replaces the current strategy of the player. 3. The opposing player executes steps 1 & 2 for themselves. 4. The whole process repeats until a stable solution is achieved. Note that the ability of fictitious play to solve a game is not guaranteed, for in some rare cases solution stability can not be achieved.



Figure 1 - Game tree representation of One-Card One-Round Poker

2.2.2 Solution to Simple Poker Variants Using Fictitious Play

Prior to a large attempted solution using fictitious play, very basic poker games were constructed and solved, one of which is presented here:

One-Card One-Round Poker (Figure 1):

- 1. Each player places 1 chip in the pot.
- 2. Each player is then presented with one card from a (in this case 169 card) deck.
- 3. One player is designated to start the betting (Player 1).
- 4. Player 1 makes a decision to bet, check, or fold.
- 5. Player 2 then makes a decision to bet, check/call, or fold.
- 6. The game ends when any player calls the other's bet, or when any player folds. The maximum number of raises is 4.
- 7. After the betting sequence, the player not to fold, or the player with the higher card wins the pot.

This simple poker variant was solved using fictitious play, the solution of which is presented in Figure 2 with the terminal node connections removed.



Figure 2 - Optimal solution to One-Card One-Round poker.

3 Abstractions

3.1 Nature of the Problem of Poker

The game of 2-player Texas Hold'em is a problem of size $O(10^{18})$ [2]. The sheer size of the problem makes clear the intractability of computing a perfect solution to the game, however there are several methods available that allow a reduction from $O(10^{18})$ to size $O(10^{7})$. Though the problem has been reduced by a significant margin, its key properties can be preserved through appropriate abstraction, and a pseudo-optimal solution can be solved for this smaller, more manageable problem.



3.2 Game Tree Abstraction

The purpose of using abstraction is to reduce the size of the solution algorithm without modifying in the underlying nature of the problem. In poker, there are available several methods of abstraction which do not detract at all from the solution, two of which are *position isomorphs* (for the two cards in the hand, or the three cards in the flop, position does not matter), and *suit equivalence isomorphs* (4s 5h in the hand preflop is equivalent to having 4c 5d, et al.). Using the available *perfect isomorphs* unfortunately does not reduce the game size to a significant enough degree to yield the problem solvable given current techniques.

3.2.1 Bucketing/Grouping

Bucketing is an excellent and commonly used abstraction technique that incurs minimal loss of information, and yields large reduction in problem size. By grouping hands of similar value into buckets, we can abstract entire groups of hands that can be played similarly into a single quantum. This method is conceptually very similar to using perfect isomorphs, for example, in our algorithm all 311 million possible flop states are bucketed into 256 categories; hands such as: "2h 4d, 3c 5s 6s" (hand, table) and "2d 5c, 4h 6d 3h" would be bucked together, and treated as having the same state. The algorithm's preflop domain contains 169 buckets, and each of the three following round domains contain 256 This is a significant improvement from buckets. previous solutions which used at most 6 or 7 buckets to describe each domain [2].



3.2.2 Chance Node Elimination

For fictitious play to be a viable solution method, a well-defined, tractable game tree must be established. In the pure solution (not abstracted), the tree can be represented as 4 tree domains (preflop, flop, turn, and river), each of which having 10 nodes (with exception of preflop because of special game rules, having 8) (refer to Figure 1 for an example of a single-domain tree). As a leaf node from one domain proceeds to the next domain (through a check/call action in anything but the root of the domain), depending on the domain change, thousands of subtrees must be formed from the movement (Figure 3). This structure becomes quickly unusable, for as the tree continues to expand through chance nodes between domains, its size increases at a rapid exponential rate.

In the solution presented, the problem regarding the exponential blow-up of the game tree size is addressed by eliminating chance nodes between domains completely. Though removal does introduce error, the essence of the chance nodes are preserved and replaced by conversion matrices which provide similar function while reducing the exponential blowup of the game tree (Figure 4). Using this strategy, each leaf node has exactly one sub domain tree associated with it. This produces a considerably lighter game tree to solve, and with the chance nodes removed from the solution, the problem is reduced to solving a tree with 6468 decision nodes instead of quite literally millions of billions of nodes.

| P (a) | := | P (a A) | P (a B) | P (a C) | P (a D) | P (A) |
|-------|----|-----------|-----------|-----------|-----------|-------|
| P (b) | | P (b A) | P (b B) | P (b C) | P (b D) | P (B) |
| P (c) | | P (c A) | P (c B) | P (c C) | P (c D) | P (C) |
| P (d) | | P (d A) | P (d B) | P (d C) | P (d D) | P (D) |
| | | - | | | - | |

Figure 5 - Progression from Domain 1 with possible states {A,B,C,D} to Domain 2 with possible states {a,b,c,d} using a conversion matrix *Assumptions:* P(A)+P(B)+P(C)+P(D) = 1P(a|x)+P(b|x)+P(c|x)+P(d|x) = 1

3.2.3 Transition Probabilities

Since the chance nodes have been completely removed from the game tree, and a bucketing approach is being used to represent states within each domain, a method to convert buckets from a *Domain A* to equivalent buckets in a *Domain B* requires a series of transition probabilities. This process can be accomplished with a conversion matrix, where each column represents a bucket within Domain A, and each row represents the corresponding probability that the Domain A bucket will translate into the Domain B bucket represented by the column number (Figure 5). These conversion matrices are expensive to compute, as each must be representative of the thousands of chance nodes which they replace.

3.2.3.1 Masking Transition

The first method explored, and one which proved to work fairly well, was to create generic transformation probabilities, convert the buckets from domain to domain, and then mask the converted bucket probabilities based on specific information about the game-state of *Domain B* (Figure 6). This method produces a generic conversion that while being imperfect, shares the same statistical properties as a tailor-made conversion distribution based on the specific game state. This method also has the benefit of being faster to calculate on the fly, requires less before-hand calculation, and requires less memory overhead than the *perfect transition* discussed next.

3.2.3.2 Perfect Transition

By calculating before-hand for every possible game-state its corresponding bucket, it is possible to use this information on the fly to create a tailor-made conversion matrix based on the game-state of Domain A, and the game-state of Domain B. This approach offers a great advantage in that it allows 'perfect' transitions between domains rather than a convincing generic transition offered by a masking method. The first of three issues invited by using this method is that since there are so many possible game states, the matrix must be created on the fly; reserving the computational resources required to create these matrices slows down calculation considerably. The second issue is that the buckets for each game-state must be known in advance (a task which depending on the problem size can easily be intractable). The last issue is that (in the case of the current implementation) the buckets for these hundreds of millions of states, once calculated, must reside in memory; using the data off the hard drive at this time seems to be an unattractive option for speed concerns.

4 Training an Optimal Player

Our algorithm which we refer to as Adam, was trained using a technique based on Fictitious Play (section 2.2), described earlier; the premise behind the training is that if two players who know everything about each other's playing style adapt their own styles long enough, their playing decisions will approach optimality. This optimality is achieved in Adam by subjecting the decision tree to randomly generated situations, analyzing how to play 'correctly' for the specific situation (based on how we know we will play and our opponent will play), and adapting the generic solution slightly toward the correct action just discovered. To solve two-player Texas Hold'em, hundreds of thousands of iterations of this basic procedure need to be applied to every node of the decision tree before the solution suitably approaches optimal play.

5 Playing Adam

Game theoretic solutions are distinguished in that the strategies produced are randomized mixed strategies; however though Adam is pseudo-optimal, the strategies produced are not entirely mixed. The decision tree represents a trimmed version of the optimal decision tree (one which would include all chance nodes between domains). Because of the abstraction chosen, the flop, turn, and river domains do not have a direct relationship with their optimal cousin; however the *preflop* domain remains unchanged even after the abstraction. This dissimilarity between domains translates into different approaches toward using the game tree in actual game play. In evaluating *preflop* states, *Adam* is able to rely on its *preflop* solution to provide approximate gametheoretic optimal strategies, and in turn, Adam uses the mixed strategies developed for *preflop* within its game play. Post-flop, Adam can not rely on the generic strategies developed through training to be suited for current board conditions, and must use them solely for reference to estimate future actions. Adam, given current information then queries the sub tree from the current decision node, and chooses the action which is assessed to have the highest value.

6 Experimental Results

Adam was played in competition against two algorithms created by a team of researchers from the University of Alberta: PsOpti, a pseudo-optimal solution, and Vexbot, a maximal algorithm. This research group has released a software package called *Poker Academy*. A pseudo-optimal solution for Two-Player Texas Hold'em was generated by our algorithm was placed in competition with both PsOpti and Vexbot via interfaces provided in their software. Figures 6-8 illustrate the results of the competition.



Figure 6. 20,000 hand performance against PsOpti. (winnings on the y-axis, hands played along x-axis)



Figure 7. 20,000 hand performance against PsOpti. (winnings on the y-axis, hands played along x-axis)



Figure 8. 50,000 hand Performance against Vexbot. (winnings on the y-axis, hands played along x-axis)

Both Figure 6 and Figure 7 represent separate competitions against another pseudo-optimal opponent. That our solution is able to consistently win against this player suggests that the solution generated by our algorithm is significantly closer to true optimality.

Figure 8 represents a competition between our derived solution, and a maximal algorithm which is designed to find flaws in pseudo-optimal solutions. The chart indicates that the maximal algorithm was unable to find faults in our solution, and therefore consistently looses as the competition progresses. This does not suggest that there are no flaws in our solution; rather that the flaws are so small that the maximal opponent is unable to detect and exploit them.

7 Future Work

As processing power and memory capacity increases, the abstractions used can be slowly weaned from the problem, and more precise solutions may be derived. We believe that increasing the current 256 buckets per round will not yield substantive benefits; rather an approach that does not totally eliminate chance nodes, but replaces extensive chance nodes (such as *preflop* to *flop* with 117 thousand branches) with a smaller group of abstracted 'bucketed' branches may lead to solutions far closer to optimality.

8 Conclusion

The expansion beyond minimax approachable games such as Chess and Backgammon has taken computer science and game theory into new areas of research. However, these new problems require different methods of solution then perfect information games, and presented is one such method applied to a domain representative of many real-world problems. Using proper abstraction techniques it is shown that fictitious play can and succeed in approaching Nash Equilibria in complex game theoretical problems such as full-scale poker.

Acknowledgements

Thanks go to Dr. C.-C. Chan for helping me publish this work. Thanks are also extended to the Department of Computer Science at the University of Akron for supporting my research efforts.

References

- Brown, G.W. Iterative Solutions of Games by Fictitious Play. In Activity Analysis of Production and Allocation, T.C. Koopmans (Ed.). New York: Wiley.
- [2] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. *Approximating Game-Theoretic Optimal Strategies for Full-scale Poker*. Proceedings of the 2003 International Joint Conference on Artificial Intelligence.