COEVOLUTIONARY NASH IN POKER GAMES

Frans Oliehoek [a] Nikos Vlassis [a] Edwin de Jong [b]

[a] *Informatics Institute, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands, {faolieho,vlassis}@science.uva.nl*
[b] *Institute of Information and Computing Sciences, Utrecht University, PO Box 80.089, 3508 TB Utrecht, The Netherlands, dejong@cs.uu.nl*

Abstract

We address the problem of learning good policies in poker games. The classical game theoretic approach to this problem specifies a Nash equilibrium solution, i.e., a pair of secure mixed (randomized) policies. We describe a new approach for calculating such secure policies based on coevolution. Here, populations of pure policies for both players are simultaneously evolved by repeated comparisons against each other, and secure mixed policies are computed from both populations by linear programming. The search heuristic for adding new candidate pure policies involves computing a best-response pure policy (by solving a POMDP) that provides a worst-case payoff for each mixed policy. We provide experimental results suggesting that a Nash equilibrium policy can be approximated in relatively few iterations, thereby producing mixed policies with relatively small support. We conclude that this is a promising direction of research and provide directions for future work.

1 Introduction

Poker games are games of partial (or imperfect) information with chance moves. The goal in such partial information games is taking optimal decisions under uncertainty. That is, the player cannot tell the exact state of the game, and he does not know what policy his opponent will follow. In poker this includes comparing potential reward to the risk involved, trying to deceive the opponent by bluffing, dealing with unreliable information from the opponents' actions, and opponent modeling, potentially for multiple players.

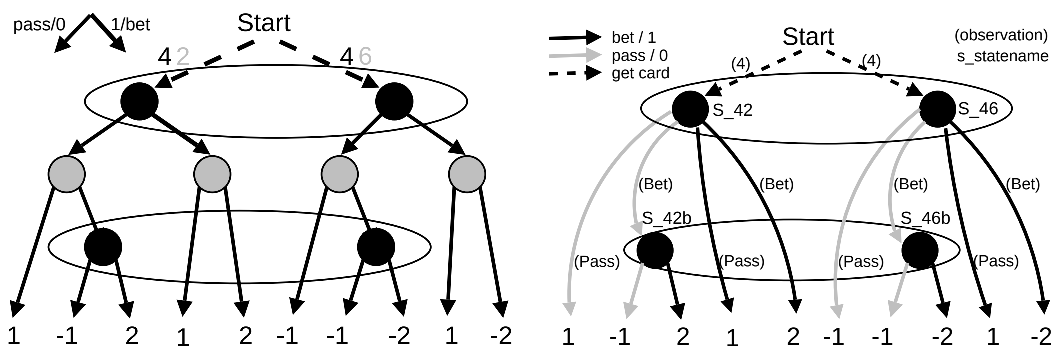
Partial information games can be represented by their game-tree or *extensive form* [4]. In this representation the players have decision nodes at which they can take actions. In addition, there are also nodes for chance moves. The decision nodes between which a player cannot discriminate are grouped in *information sets.* An example of an extensive form game is given in figure 1a. A *pure policy* for an extensive form game is a mapping from information sets to actions.

The game theoretic solution for such games is a *Nash equilibrium*, i.e., a pair of policies that are a best-response to each other. By using a Nash equilibrium policy ('Nash policy'), a player guarantees himself a security level payoff. In general, a Nash equilibrium might not exist in pure policies. Instead the solution must be sought in convex combinations of pure policies (*mixed policies*) or in *stochastic policies*: mappings from information sets to probability distributions over actions.

The standard method for solving two-player zero-sum games like poker is *linear programming* (LP) [12, 9]. Although there are efficient (polynomial-time) algorithms for LP, their solutions become expensive for large problems. A challenging topic is therefore how to compute (approximate) Nash equilibria with smaller linear programs or even without using linear programming.

A method for finding Nash equilibria in poker using LP and stochastic policies is presented in [3]. The described system, *Gala*, uses a *sequence form* representation that is polynomial in the size of the game-tree. Still, for real-life poker variants the game-tree can be very large. Existing methods for reducing the complexity of Gala are based on abstraction or binning [10, 1].

In this paper we propose a complementary approach to efficiently calculate Nash equilibria for poker games based on coevolution [2]. In this approach the single LP as used in Gala is replaced



(a) Gambler's decision nodes are black, dealer's are gray.

(b) Decision nodes become states.

Figure 1: A partial game-tree for 8-card poker (a) and the corresponding POMDP model for the gambler (b). Both figures show the deals $(4, 2)$ and $(4, 6)$. The ovals the indicate how the decision nodes the gambler can't discriminate between are grouped in information sets.

by multiple smaller LPs. In this paper we will focus on small poker games; as example we will use the *8-card poker*, a small poker game from literature [3].

In section 3 the approach for calculating best-response policies is described. Next, in section 4 we describe the Nash memory mechanism from [2] and how this approach is extended to deal with poker games. Experimental results are given in section 5. Section 6 concludes and gives directions for future research.

2 8-card poker

This poker variant is played by two players: a dealer and a gambler, who both own two coins. Before the game starts, they pay one coin to the pot, the ante. Then both players are dealt one card out of a deck of eight cards (1 suit, ranks 1–8). After the players have observed their card, they are allowed to bet their remaining coin, starting with the gambler. If the gambler bets his coin, the dealer has the option to fold or call. If the dealer folds he loses the ante. If he calls, showdown follows. If the gambler didn't bet, the dealer can choose to bet his coin. If he does, the gambler will have to decide whether to fold or call. If the game reaches the showdown (neither player bets or the bet is called), the player with the highest card wins the pot.

Figure 1a shows a partial game-tree for 8-card poker. The figure contains branches for two deals; in the left branch the dealer receives card 2, in the right branch he receives card 6. The gambler receives card 4 in both cases. Because of this, the gambler can't discriminate between the displayed decision nodes. As a consequence these nodes are grouped in information sets. The outcomes shown are the payoff for the gambler.

3 Best-response play

As shown in [6], when the stochastic policy of the opponent is fixed and known (e.g. estimated from repeated play), the poker game can be recast as a partially observable Markov decision process (POMDP) [11] from the perspective of the protagonist agent. Solving this POMDP results in a best-response policy, giving the highest payoff obtainable against the fixed opponent policy. Here we give a brief outline of this approach.

The decision nodes for the protagonist agent together with the outcome nodes constitute the state space $\mathcal{S}$ for the POMDP. Taking a particular action, a_i, from one of these states $s \in \mathcal{S}$ leads to an opponent node t.[1] If the opponent in turn selects some action a_j this will lead to a successor state s' for the protagonist agent. Therefore, the chance of transferring to s' from s is given by:

$$P(s'|s,a_i) = \sum_{a_j} \sum_{t \in T} P(s'|t,a_j) P(a_j|t) P(t|s,a_i).$$

Because we assume that we know the stochastic opponent policy $P(a_j|t)$ and we know $P(s'|t,a_j)$ and $P(t|s,a_i)$ which follow from the description of the game, effectively, this transforms the deterministic transitions from the extensive form to stochastic transitions in the POMDP model as illustrated in figure 1.

It is known that solving a POMDP gives an optimal deterministic policy [8]. Solving a POMDP, however, is intractable in general. In the special case of poker, in which the transition model has a tree-like structure, solving the POMDP is relatively easy. Because the tree is finite, the number of possible beliefs is bounded: there is exactly one belief for each information set. Therefore all possible beliefs and transitions between them can be generated, resulting in a completely observable MDP over beliefs. This MDP can then be solved using standard dynamic programming techniques, such as value iteration [8].

4 Coevolution

We will first present a coevolutionary approach to calculating Nash equilibria ("the Nash memory mechanism") as proposed by [2]. Next we will extent this to poker.

4.1 The Nash memory mechanism

The Nash memory mechanism presents a method for iteratively reaching a Nash equilibrium for two-player, symmetric games. In a symmetric game, e.g. Rock-Scissors-Paper, both players select their (possibly mixed) policy from the same set of pure policies available for the game.

Because in symmetric zero-sum games the expectation of a policy played against itself is 0, $\forall_\pi \; E(\pi,\pi) = 0$, a Nash policy should provide a security-level payoff of 0. Let $S(\pi)$ denote the *security set* of policy π, i.e., $S(\pi) = \{\pi' | E(\pi,\pi') \geq 0\}$. We are thus searching for a policy π, such that $\forall_{\pi'} \; \pi' \in S(\pi)$.

Let $\mathcal{N}$ and $\mathcal{M}$ be two mutually exclusive sets of pure policies. $\mathcal{N}$ is defined to be the support of mixed policy $\pi_\mathcal{N}$, which will be the approximation of the Nash policy during the coevolution process. The set $\mathcal{M}$ will contain encountered policies that are not needed by $\pi_\mathcal{N}$ in order to be secure against all encountered policies, i.e. $\mathcal{N} \cup \mathcal{M} \subseteq S(\pi_\mathcal{N})$.

Apart from these two sets, the Nash memory mechanism specifies a search heuristic H. This is an arbitrary heuristic that delivers new tests against which $\pi_\mathcal{N}$ is evaluated.

To start, $\mathcal{M}$ is initialized as the empty set and $\mathcal{N}$ is initialized as a set containing an arbitrary pure policy. $\pi_\mathcal{N}$ is initialized as the 'mixed' policy that assigns probability 1 to this pure policy. At this point the first iteration begins. Figure 2 shows one iteration of the Nash memory. First, a set of test-policies, $\mathcal{T}$, is delivered by the search heuristic and evaluated against $\pi_\mathcal{N}$, to define the set of 'winners':

$$\mathcal{W} = \{\pi \in \mathcal{T} \,|\, E(\pi, \pi_\mathcal{N}) > 0\}.$$

When this set is non-empty, clearly $\pi_\mathcal{N}$ is not a Nash equilibrium strategy, as it is not secure against all policies. Therefore $\pi_\mathcal{N}$ should be updated in this case.

First a payoff matrix of all policies in $\mathcal{M} \cup \mathcal{N} \cup \mathcal{W}$ played against each other is constructed. In this matrix, the rows represent the pure policies of the first player and the columns those of the second player. Matrix entry (i,j) gives the payoff of policy i played against policy j. Together with the constraint that the probabilities of a mixed policy must sum to 1 and the desire of both players to maximize their own payoff, this matrix can be converted to a LP, which then can be solved using linear programming. The result will be a new policy $\pi'_\mathcal{N}$, the policies to which it assigns positive weight, $\mathcal{N}'$, and the other policies, $\mathcal{M}'$. At this point a new iteration is started. The whole process is repeated until convergence.

[1]In the presence of chance moves, this is a probability distribution over opponent nodes.

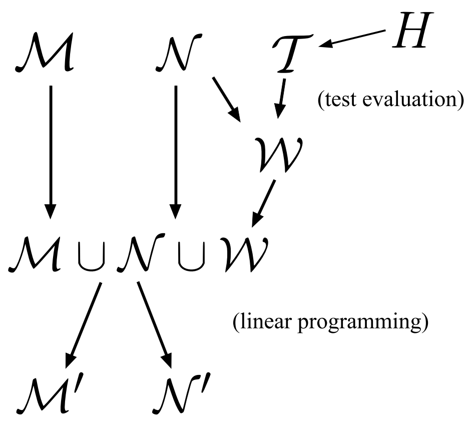


Figure 2: One iteration of Nash memory coevolution.

4.2 Poker games

In order to apply the procedure outlined above on poker games, a suitable search heuristic and a way to generalize to asymmetric games is needed. A simple solution for the latter would be to create a new compound game consisting of two games of 8-card poker: one played as gambler and one played as dealer. This compound game is symmetric and a particular policy π^i is given by $\pi^i = \left\langle \pi^i_{gambler}, \pi^i_{dealer} \right\rangle$. Using this representation the Nash memory mechanism can directly be applied without changes. However, it is clear that the flexibility, which the new mixed policy is constructed with, is constrained.[2] Therefore we use a different approach.

Note that in the procedure above, there are only two reasons why the game must be symmetric: to determine whether a test policy $T \in \mathcal{T}$ beats the current mixed policy, $E(T, \pi_{\mathcal{N}}) > 0$, and because the next Nash approximation is calculated from one large set containing all encountered policies (the set $\mathcal{M} \cup \mathcal{N} \cup \mathcal{W}$).

The idea behind our approach is to apply the Nash memory mechanism per player, i.e we maintain sets $\mathcal{M}_p, \mathcal{N}_p, \mathcal{T}_p, \mathcal{W}_p$ and a Nash approximation $\pi_{p,\mathcal{N}}$ for both players $p = 1, 2$. Now, if, without loss of generality, we assume that the search heuristic delivers a single test policy for both players, T_1 and T_2, we can test whether the compound policy $T = \langle T_2, T_1 \rangle$[3] beats the compound policy $\pi_{\mathcal{N}} = \langle \pi_{1,\mathcal{N}}, \pi_{2,\mathcal{N}} \rangle$, as:

$$E(T, \pi_{\mathcal{N}}) = E(T_2, \pi_{2,\mathcal{N}}) + E(T_1, \pi_{1,\mathcal{N}}).$$

If $E(T, \pi_{\mathcal{N}}) > 0$, then $\pi_{\mathcal{N}}$ is not secure against T and the policy should be updated. In this case let $\mathcal{W}_1 = T_2$ and vice versa. This results in two sets $\mathcal{M}_1 \cup \mathcal{N}_1 \cup \mathcal{W}_1$ and $\mathcal{M}_2 \cup \mathcal{N}_2 \cup \mathcal{W}_2$ of pure policies for respectively gambler and dealer. By constructing the payoff matrix for these pure policies and applying linear programming we calculate $\pi'_{1,\mathcal{N}}$ and $\pi'_{2,\mathcal{N}}$, finishing one iteration.

Next, we turn our attention to the search heuristic. This is an important aspect for coevolutionary approaches as it should be powerful enough to discover improvements to the current candidate solution, i.e., it has to find policies that beat the current Nash approximation. The approach outlined in section 3 provides a suitable candidate, as a best-response policy obtains the highest payoff possible. This also means that it provides the worst case payoff of the current Nash approximation. Another nice effect is that this provides a convergence criterion: when the best-response policies do not attain a positive payoff in the compound game, $E(T, \pi_{\mathcal{N}}) = 0$, then $\pi_{\mathcal{N}}$ is a Nash policy.

However, using the approach from section 3, it is possible to calculate a best response against a stochastic policy. In contrast, the Nash approximations, are mixed policies, making it necessary to convert such a mixed policy to a stochastic policy. Proof that this is possible and the accompanying algorithm is given in [5]. The intuition is the following. We want to know the probability $P(a|I)$ of an action a at an information set I that corresponds with a particular mixed policy μ. Every

[2]Observe that it is not possible to put more weight on a particular gambler policy $\pi^i_{gambler}$ without putting the same weight on the corresponding dealer policy π^i_{dealer}.

[3]Note that a test policy T_1 for player 1, is a policy for his opponent, player 2, and vice versa.

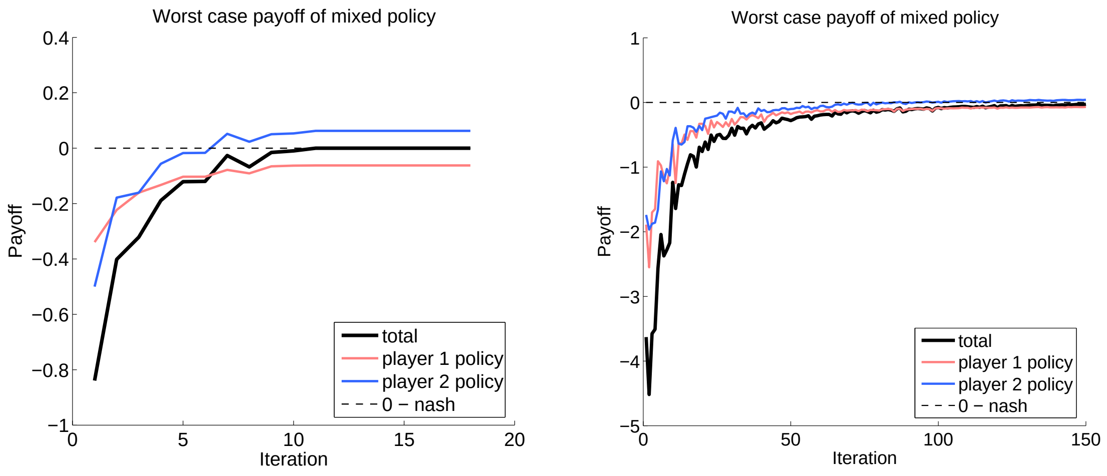


Figure 3: Results for the Nash memory approach. Left: 8-card poker. Right: 2-round, 3-bets-per-round 6-card poker.

pure policy π in μ's support only allows a subset of information sets to be reached. Therefore at I, π's contribution to $P(a|I)$ has to be weighted by the chance that this policy, π, was responsible for realizing I.

5 Experiments

The procedure described in section 4.2 was implemented and applied to 8-card poker. Figure 3 shows the results. It only takes a few iterations to obtain a policy that is fairly secure, implying that this technique might be applied for larger games to obtain an approximate Nash policy.

The figure also indicates that only a relatively small number of pure policies are needed to make up a secure mixed policy. It turns out that the number of pure policies used by the mixed policy is even lower than the figure suggests: when reaching the Nash level (iteration 12) only 6 out of 12 pure policies are assigned weight for the both gambler and dealer policy.

Although convergence to Nash equilibrium is guaranteed [2], we can observe that the worst case payoff does not increase monotonically. Even though with every iteration the approximate Nash becomes secure against more policies, a particular policy against which it is not secure yet might become a best-response and do more damage than the current best-response.

5.1 Some larger poker games

After the encouraging results for 8-card poker some experiments were performed on larger poker games. The results of these were similar, therefore we restrict our discussion to one of them. It is a 2 round poker game with a deck of 6 cards, both players receive one card and play a bet-round, after which 1 public card appears face-up on the table. Then a final bet-round is played. In both bet-rounds a maximum of 3 coins can be bet per player. The game-tree for this game consists of over 18,000 nodes.

The obtained results are shown in figure 3. As was the case for 8-card poker, the Nash memory is able to obtain a reasonable security level in a relatively low number of iterations. The small number of policies needed for the support of the mixed policy was also confirmed for the larger games: at iteration 150 for the 6-card poker game[4], the number of policies with positive weight was 29 for both players.

[4]The algorithm was not fully converged at this point, as it still received a worst case payoff of -0.027 instead of 0.

6 Discussion and conclusions

When comparing the performance of the approach as given in this paper against solving sequence form as in [3] there are some interesting differences. At the moment, the Nash memory approach uses more time compared to sequence-form solving. However, it spends its time differently: most is spent during constructing and solving the POMDP models and determining outcomes between the encountered pure policies. Far less time is spent on linear programming, as the size of the linear programs to be solved is generally smaller: e.g. the 6-card poker game contains 2162 sequences for both players and therefore requires solving a linear program with a payoff matrix of size 2162×2162. In contrast, the largest linear program solved by the Nash memory approach, has a matrix of size 150×150 for the same problem.

Also, we expect that considerable speed-up can be obtained by streamlining the implementation of POMDP model construction and solving. Moreover, approximate methods could be used for both solving the POMDP and evaluating the rewards. This might make this approach competitive in terms of performance with respect to sequence form solving.

Currently we are investigating whether the coevolutionary approach might be useful in making a tradeoff between the best-response and security level payoff.

Another direction of future research would be to try to avoid solving a linear programming from the start in each iteration. There might be an approach to adjust the weights of the mixed policy without solving a complete linear program.

A final point is to focus on extending this approach to multiple players. A form of 'symmetrization' might also be possible in this case. Finding a secure mixture of policies could be done using any of the methods described in [7].

References

[1] Darse Billings, Neil Burch, Aaron Davidson, Robert Holte, Jonathan Schaeffer, Terence Schauenberg, and Duane Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *Proc. Int. Joint Conf. on Artificial Intelligence*, Acapulco, Mexico, August 2003.

[2] Sevan G. Ficici and Jordan B. Pollack. A game-theoretic memory mechanism for coevolution. In *GECCO*, volume 2723 of *Lecture Notes in Computer Science*, pages 286–297. Springer, 2003.

[3] Daphne Koller and Avi Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1-2):167–215, 1997.

[4] H.W. Kuhn. Extensive games and the problem of information. *Annals of Mathematics Studies*, 28:193–216, 1953.

[5] Frans Oliehoek. Game theory and AI: a unified approach to poker games. Master's thesis, University of Amsterdam, 2005.

[6] Frans Oliehoek, Matthijs T. J. Spaan, and Nikos Vlassis. Best-response play in partially observable card games. In *Benelearn 2005: Proceedings of the 14th Annual Machine Learning Conference of Belgium and the Netherlands*, pages 45–50, February 2005.

[7] Ryan Porter, Eugene Nudelman, and Yoav Shoham. Simple search methods for finding a nash equilibrium. *Games and Economic Behavior*, (to appear).

[8] M. L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., New York, NY, 1994.

[9] R Tyrrell Rockafellar. *Convex analysis.* Princeton, N.J., Princeton University Press, 1970.

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

[11] E. J. Sondik. *The optimal control of partially observable Markov decision processes.* PhD thesis, Stanford University, 1971.

[12] J. von Neumann and O. Morgenstern. *The Theory of Games and Economic Behavior.* Princeton University Press, 1947. 2nd edition.