Approximating n-player Behavioural Strategy Nash Equilibria Using Coevolution

Spyridon Samothrakis
School of Computer Science
University of Essex
Wivenhoe Park, Colchester, CO4 3SQ
CO4 3SQ
United Kingdom
ssamot@essex.ac.uk

Simon Lucas
School of Computer Science
University of Essex
Wivenhoe Park, Colchester, CO4 3SQ
CO4 3SQ
United Kingdom
sml@essex.ac.uk

ABSTRACT

Coevolutionary algorithms are plagued with a set of problems related to intransitivity that make it questionable what the end product of a coevolutionary run can achieve. With the introduction of solution concepts into coevolution, part of the issue was alleviated, however efficiently representing and achieving game theoretic solution concepts is still not a trivial task. In this paper we propose a coevolutionary algorithm that approximates behavioural strategy Nash equilibria in n-player zero sum games, by exploiting the minimax solution concept. In order to support our case we provide a set of experiments in both games of known and unknown equilibria. In the case of known equilibria, we can confirm our algorithm converges to the known solution, while in the case of unknown equilibria we can see a steady progress towards Nash.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Plan execution, formation, and generation*

General Terms

Algorithms, Performance, Experimentation

Keywords

Coevolution, Nash Equilibrium, Tree Searches

1. INTRODUCTION

Artificial Coevolution [21], roughly speaking, is a process where one agent improves relative to another set of agents, hopefully in a never-ending arms-race [6]. There is a number of reasons why one might prefer using coevolution rather than regular artificial evolution. The major one is that the environment adapts to the agent, making any effort to adapt versus a snapshot of that environment futile. This is typical in multi-agent settings. For example, if one wanted to find an optimal tennis agent (player), the only available and optimal evolutionary solution would be to somehow evolve against an optimal agent, which is logically self-defeating (in the sense that if we already have an optimal agent, there may be little point in evolving one). In coevolution, one would start with a selection of players which would try to compete against each other, ideally reaching good performance after some evolutionary cycles. Another potential problem with the standard evolutionary (as opposed to coevolutionary) approach would be that the environment is too hard to evolve against, and therefore one can hope that coevolution will provide a smoother learning curve. There is ample evidence from artificial life and biology (for example [26]) that learning it easier when the difficulty of the task progresses in an incremental fashion. By trying to evolve against an optimal agent/environment we run the risk of creating a situation where we never get any meaningful gradient in order to warrant any progress. In practice, there is often a simple solution to this latter problem of reduced gradient by simply making a strong opponent weaker by forcing it to make random moves.

Unfortunately, due to a set of issues, the situation is almost never ideal in coevolutionary domains. The evolved agents are usually stuck in suboptimal/mediocre solutions or cycle in the solution space, never actually reaching a fully optimum strategy (ex. see [5, 27]). This problem has been identified in a series of publications (see next section), and it is directly related to optimality - in other words when do we consider an agent optimal in a coevolutionary setting, so that we get a gradient towards that solution. The major contribution of this paper is a novel method for what we think is the most popular solution concept, Nash equilibria in n-player zero sum games. Compared to an alternative method of finding Nash equilibria using coevolution proposed by Ficici [9] (see [17] for a real game implementation), our algorithm does not require either an archive nor solving linear programming equations; the end result of the algorithm is just one agent, rather than a collection of agents; we also think there are certain advantages we have compared to non-evolutionary methods, mostly related to imperfect recall (i.e when the agent forgets parts of the historical events that took place in a game).

The rest of this paper is organised as follows: Section 2 provides the necessary background in coevolution and game theory. Section 3 describes our algorithm. In Section 4 we perform a number of experiments where we showcase the efficiency of our new algorithm in a set of games. Finally, we provide some insights and conclude this work at Section 5.

2. BACKGROUND

2.1 Evolution & Coevolution

In traditional artificial evolution each individual g_i in a population of individuals G is transformed to its corresponding phenotype and assigned a value $F : X \rightarrow R$, $G \subseteq X$ where function F is termed the *fitness function*, and X is the search space. A set of operators is applied on the population (ex. mutation, crossover), and we move to the next generation. The process continues until some stopping criterion is met. Coevolution does almost the same thing, with the exception that fitness is measured relative to other agents. Thus, at least in competitive coevolution, we are trying to evolve agents in multi-agent settings. This creates a situation where there is no *objective* function to measure overall progress, but rather a series of *subjective* tests [19]. In a quite large number of cases it leads to mediocre performance from the evolved agents, mostly due to intransitivity [7]. Agent A is able to outperform agent B, and agent B outperforms agent C, but it might turn out that C is better than A.

2.2 Single Agent Environments and Evolution

2.2.1 Markov Decision Process

Most policy searching problems can be conceptualised as a Markov Decision Process (MDP). The MDP is a tuple $(\mathcal{S}, \mathcal{A}, T, f)$ where: $\mathcal{S}$ is a set of states, $\mathcal{A}$ a set of actions, $T(s'|s,a)$ is the probability of moving from state s to s' after action a, $f(s,a)$, a reward function at each state (although in practice lot's of states might have one that is 0). The MDP environment is the most commonly used environment when evolving agents. It assumes that we have a single agent environment, fully observable to the agent. One can also think of that as a singe player game. A learner's goal in this setting is to learn a controller, or a policy, that maximises reward. Evolution is used a *policy search* method, tweaking some vital parameters of a controller.

2.2.2 Partially Observable Markov Decision Process

A Partially Observable MDP (POMDP) [15] is described by a tuple $P = \langle \mathcal{S}, \mathcal{A}, T, f, O, N, b_0 \rangle$. $\mathcal{S}$ are the world states, $\mathcal{A}$ is a finite set of actions available to the agent, T is a transition function, so that $T(s'|s,a)$ is the probability of moving from state s to s' after action a. $f(s,a)$ is a reward function. O is a finite set of observations. $N(a|s',o)$ is an observation function which denotes the probability of receiving observation o if the agent applies action a and the world moves to state s'. b_0 is the agent's initial belief, such that $b_0(s)$ is the probability that the initial state is s. In the POMDP setting, an agent needs to encode historical actions in the environment as well, as he is having *beliefs* over possible states. Again, one can use evolution in this setting as well, and beliefs over states can either be explicitly given to the agent (transforming the POMDP to an MDP) or hope that the evolutionary process will uncover them.

2.2.3 Multi-Agent Environments and Coevolution

The MDP and POMDP formalisms described above cannot be mapped directly to multi-agent settings, as they only account for single agent dynamics. One has to look to game theory in order to find suitable representations for multi-agent problems. Broadly speaking a game[1] can be defined as the tuple $\langle N, O, \sigma, f_{\mathcal{A}}, (\mathcal{I}_i)_{i \in N} \rangle$, where N is the set of players, O are sequences of actions α_k of each player (Histories) drawn of $\mathcal{A}$, $\sigma(h)$ a is a function that takes a history and outputs and action, $f_c(\alpha|h)$ gives the payoffs, and , finally , $(\mathcal{I}_i)$, is an information set, i.e. the histories which an agent cannot distinguish. For this paper, an agent and a strategy σ are equivalent notions. There is a number of ideas that we need to map from game theory to evolution. The first one is the concept of a *strategy*. In the case of evolution, one can intuitively think of the agent, agent policy and strategy interchangeably. The second idea is the idea of a solution concept. A solution concept is what one would deem as solution to the game and it was transferred from game theory to evolution by Ficici's Ph.D thesis [9]. In normal (single agent) evolution for example, a possible solution concept would be "the individual with the highest fitness score". In coevolution there is a number of possible solution concepts (ex. Pareto dominance, Nash, Maximum expected values, see [22]), and one has to make it explicit which one is aiming at.

[1]For exact definitions of games and strategies, please see [20]

2.2.4 Nash Equilibria Solution Concept in Coevolution

The Nash equilibrium is a solution concept that states that once all players have reached a specified set of strategies, none has an incentive to deviate from his strategy unilaterally. Formally , we can define $G : (S, f_c(\alpha|h))$ as a game with n players, S_i is all possible strategies for player N_i, S is the set of all possible strategy profiles, defined as the Cartesian product between all possible strategies for each player ($S = S_1 X S_2 X S_3 ... X S_n$). The payoff function is as seen previously, $f_c(\alpha|h) = (f_1(\alpha|h), ..., f_n(\alpha|h))$. Each player can choose a strategy σ_i, which creates a strategy profile $\sigma = \sigma_1, \sigma_2, \sigma_3 ..., \sigma_n$. Let σ_{-i} be a strategy profile that lacks a strategy for player N_i. A strategy profile σ^* is a Nash equilibrium if:

$$\forall p_i, \sigma_i \in S_i, \sigma_i \neq \sigma_i^* : f_i(\sigma_i^*, \sigma_{-i}^*) \geq f_i(\sigma_i, \sigma_{-i}^*) \qquad (1)$$

In the above (eq.1), wherever one sees a strategy, it can be replaced with agent/phenotype. Although the Nash equilibria is a concept that applies to strategy profiles, i.e. collections of agents, in this paper, if a strategy if part of a Nash profile, we will call this strategy a Nash strategy.

Finding Nash equilibria might turn out to be harder than anticipated, so we alternatively should aim towards the concept ϵ-Nash [29].

$$\forall p_i, \sigma_i \in S_i, \sigma_i \neq \sigma_i^* : \epsilon + f_i(\sigma_i^*, \sigma_{-i}^*) \geq f_i(\sigma_i, \sigma_{-i}^*) \qquad (2)$$

2.2.5 Behavioural Strategies

It is not necessary that a single strategy for each player will suffice for Nash equilibria, it has however been proven [18] that a weighted probability over a number of *pure strategies*, known as a *mixed strategy*, will always exist in zero sum games as part of the Nash profile. In coevolution this has been solved by attributing a probability distribution over a number of evolved solutions [8]. In case an agent however has perfect recall, i.e. all past states can be remembered by our agent, one can attribute probabilities to each specific action [4] and create what is termed "behavioural strategies". Thus, at a specific state, if there are three possible actions, an agent might need to perform action one with 0.2, action two with 0.3 and action three with 0.5 probability in order to be part of Nash profile.

One might question the ability of a real organism to perform mixed strategies; it is not often that we think we might base a decision on probabilities. The primary motivator for using mixed strategies it to avoid becoming predictable. Once you become predictable, an opponent has a strong incentive to switch to a strategy that exploits your traits. If however you maintain the right balance between moves (while in Nash Equilibria), modelling you is not beneficial any more and any effort of your opponents to actively exploit one of your models will probably lead to bad performance from their part. Behavioural Strategies do have biological motivation; for example in [12] one can see that animals explicitly mix strategies, not just at a population level, but at personal level as well. Finally there is a whole field of study dedicated to studying human decision making at a neuronal level [14].

The advantage of using behavioural strategies rather than mixed strategies has to do with the complexity of the problem. Even in a trivial to solve game like one card poker with 13 cards, one has 2^{26} pure strategies, which means at best we have to evolve that many strategy weights. On the other hand, using behavioural strategies we only have to search to search space of $26 * 4 * 3$ real numbers in order to find a possible solution[2].

2.2.6 *Zero Sum Games, Minimax and Nash*

Intuitively, once a strategy profile (where all the strategies are the same) is formed, one can measure a best response (BR) strategy against the profile by calculating a pure counter strategy. We can then modify our strategy so it can minimise the expect reward of the $BR(\sigma_i)$ strategy, and keep doing that process iteratively until $BR(\sigma_i)$ has approached zero expected reward (the player is not exploitable). Another way to see this is that we are looking for an agent that can solve the minmax problem (eq. 3), that is, we are actively trying to minimise our opponents maximum payoff; Nash equilibrium and minimax are equivalent concepts in zero sum games (but not in general sum).

$$\arg\min_{\sigma_1 \in S} \max_{\sigma_2 \in S} f(\sigma_1, \sigma_2) \tag{3}$$

Effectively, we are looking to improve the worst case performance of our agent. The importance of this cannot be overstated, and it is the base of most successful game playing agents (ex. chess). Unless we have a clear model of our opponents at each time step, the best strategy we can afford is one where we assume the worst possible opponent.

2.3 Tree Searches

By assuming that there are two ontologically different parts of nature, other agents and the environment, and that our agent knows both about this dichotomy and that is aware of the causality of nature, we can start performing searches in the future in order to find out an optimal set of actions. The reason why we need this dichotomy is because other agents are not only observed by us, but they observe us as well. Thus they might have incentives to change their strategies based on our actions. This would in turn lead us to have an incentive to deviate from our own actions etc, ad infinitum. The process is captured in [28]. Alternatively, one can however make the assumption that the process described previously can continue for ever, so it makes sense to start thinking of your opponent as the best opponent possible. This kind of thinking has lead to wide array of algorithms that exploit the forward model mostly using tree searches, and we briefly visit them below.

2.3.1 *Minimax*

Provided a problem has perfect and complete information, i.e. there is no source of randomness in the world, all moves are observable, and a model of the world is encoded by the agent, one can use the minimax [23] algorithm in order to choose which action one should perform at each time step. The agent simulates a tree of possible future states, at each level trying to identify moves that minimise his potential loss against his opponents. The idea here is, since we don't have a model of the opponents in the world, but we do have one of nature, we must try and find a worst case solution, i.e. , the equivalent of playing against the strongest possible player. Usually, because simulating the future indefinitely is computationally expensive, the search is stopped at a certain depth and some heuristic value function is used as the reward. Of course, it would all be easier if our world model included a model of our opponents, and we could simply extrapolate to the future, but in most scenarios this model is simply not there. Thus, instead of trying to win, we actively try not to lose, hoping that mistakes in our opponent behaviour will lead to his demise. The algorithm, and variations of it, have been proved extremely successful, effectively forming the basis of a large number of game players (ex. Tron, Chess).

2.3.2 *Expectimax*

If the underlying world/game we are trying to reason in has an element of incompleteness, i.e randomness, we need some way of encoding this randomness to our future extrapolation process. Thus, an expectimax tree is the same as a minimax tree, with the exception of random event nodes. We usually introduce, a third player, explicitly termed nature, which takes turns modifying the world in a random fashion. We now aim to find moves that minimises our losses in expectation.

2.3.3 *Miximax*

Contrary to games of perfect information, in games of imperfect information none of the previous algorithms (i.e. minimax, expectimax and their Monte Carlo approximations) can be used. This is for two reasons: the first one is that there is nothing to min or max, as we lack knowledge about the current state the world. The second is that the nodes are not independent. Thus, even if we have an opponent model, and thus no need to min/max, we would end up solving a series of perfect information games (see figure 3). In order to counter this, one can actively learn how opponents will behave, group sets of states into information sets, and try to exploit this using perfect information tree searches. The opponent behaviour learning can potentially be done using any machine learning method. The resulting tree is called miximax [3] and behaves like a tree search on a POMDP (after it has been transformed on a MDP), although there is no explicit Bayesian belief update (a machine learning algorithm gives the requested rewards/beliefs).

2.3.4 *Monte Carlo Tree Searches*

The trees presented above are usually extremely long and cannot be analytically calculated, but for the smallest worlds. There are various methods of pruning the tree, and terminating the search early. Recently, a new way of approximating the tree searches has been proposed, termed "Monte Carlo Tree Search" (MCTS). In MCTS, each node in the tree is seen as multi-armed bandit [2]. The goal of the search is to "push" more towards areas of the search space that seem more promising. Although there are slightly different versions of the algorithm, the one presented in [10] is the "original" version. The algorithm can been summarised as follows; starting from the root node, expand the tree by a single node. If the node is a leaf node, back-propagate the value to the node's ancestors in the tree. If not continue expanding into deeper nodes by using a bandit strategy ex.

$$\bar{x_j} + C\sqrt{\frac{ln(n)}{n_j}} \tag{4}$$

[2]This depends to a certain extend on the encoding we use, more concerning this in the methodology section

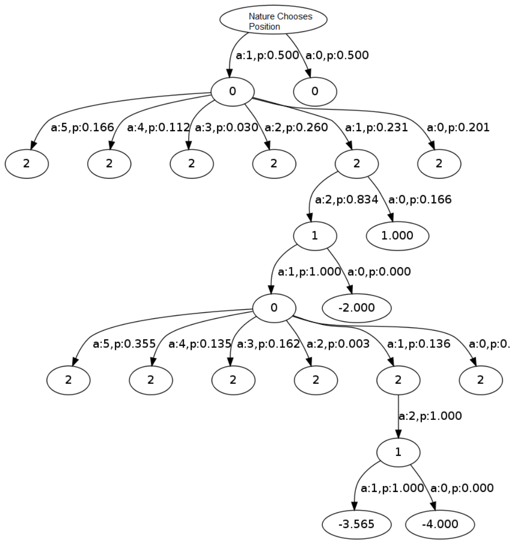


Figure 1: A sample poker tree using Bayesian updates. Observe that there is no player 2, as we have converted the game to a single player POMDP. Player two action probabilities are acquired through some machine learning algorithm.

If the node is not a leaf node, keep exploring the tree until one is reached. There have been many improvements on the algorithm above, with some modern versions barely resembling the idea originally proposed, but the concept remains the same.

3. METHODOLOGY

3.1 Evolutionary Strategies

Whenever evolution is mentioned a $(\mu, \lambda) - CSA - ES$ [11] *Evolutionary Strategy* is used, where the parents μ for each generation are always half the offspring population $\lambda = 10$. Our use of CSA-ES is mainly motived by its excellent performance in previous applications. In that respect, any evolutionary algorithm or genetic algorithm that supports a real-valued encoding could be used. In our experiments each state in the game is assigned a number of genes that are necessary to describe it (see next subsection).

3.2 Treating Behavioural Strategies

3.2.1 *Representing Behavioural Strategies*

We present two ways of representing behavioural strategies. One is by direct encoding. That is, for each tuple of histories ($a_x^{t=0}$, $a_x^{t=1}$, $a_x^{t=2}$, ..., $a_x^{t=n-1}$) (or an abstraction of histories) we can encode another tuple (o_1, o_2, o_3, o_x) which describes the probabilities for each action using equation 5:

$$p_a(k) = \frac{o_k}{\sum_{i=0}^{n} o_i} \tag{5}$$

Another representation involves using Neural Networks(NNs). NNs are a category of universal function approximators [25] loosely based on the function of biological neurons. In this paper we will concern ourselves with one of their simpler incarnations, the multi-layer perceptron. The multi-layer perceptron is an directed graph, usually set-up in an input-hidden-output layer fashion. The input vector presented to the neural network will be transformed to an output vector using equation 6.

$$f(x) = G_2\left(b_2 + W_2\left(G_1\left(b_1 + W_1 x\right)\right)\right) \tag{6}$$

In our case, G_2 is the sigmoid function and G_1 is $tanh$. The inputs i_l of the network can be histories (if we we have perfect recall) or some other measurement we acquire from the game. Thus, for each history/state we provide some inputs to the neural network (ex. past histories, current state abstractions) and we get outputs which we then feed equation 5 to turn into action probabilities.

3.2.2 *Measuring the Expected Value of behavioural Strategies*

How can one evaluate an agent that encodes a behavioural strategy? Even if there is no source of randomness in the environment, we introduce such randomness by having our agents take actions at random. Thus, there are two ways of measuring the expected value of our agents. The first one would be pit the agents against each other and keep playing games repeatedly for a number of iterations, until we are satisfied that we have found the expected value. This might even be necessary in case where the rewards, actions and states are continuous as we might have trouble properly discretizing the resulting spaces. However if we can somehow get past this, we can easily treat the game as an expectimax tree without the min nodes. Instead of min nodes, we can now plug in the probability of each action at each state/information set. This will result in very fast and exact measurement of the expected values of all involved agents, with the drawback of a much larger memory consumption.

3.3 Purely Evolutionary Solution

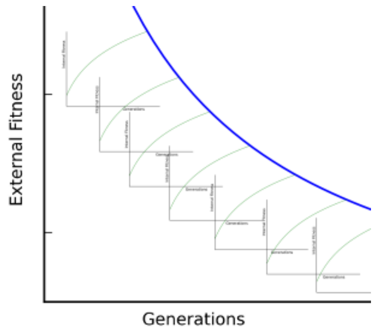


Figure 2: Imaginary progress of the purely evolutionary version of the algorithm. The external evolutionary run tries to approach 0 (not exploitable by anyone), while the internal evolutionary runs try to maximise their maximum score, thus providing "scaffolding" (and a gradient) for the external run

We will present two methods of reaching Nash Equilibria. The first one involves using just evolution. Both algorithms are presented in Algorithm 1. For the purely evolutionary version, in order to evaluate each genome, we run a contra-evolutionary run, trying to find out exactly the worst possible performance of the genome. In short, the fitness of each genome in the outer loop is evaluated by having someone evolve an agent against it in the inner loop. The first heuristic that we use here is that we provide the inner loop a good starting point, which in symmetric games would be either the best candidate from the outer loop or the best candidate from the previous run of the inner loop. Furthermore, it makes sense to increase the number of generations in the inner loop as we progress through the generations of the outer loop. Initially, we can accept weak or mediocre response strategies, whereas in later stages we need almost optimal agents in order to measure any meaningful best response. In figure 2 we can see such an evolutionary run .

Algorithm 1 $minmax - coevolve()$

```
function $coevolve()$
Initialise Evolutionary Run
$k_{ou} \leftarrow$ The outer loop generations
$k_{in} \leftarrow$ The inner loop generations
for l $\leftarrow$ 0 to $l = k_{ou}$ do
  for each $G_{ij}$ in $G$ do
    if $pureCoevolution$ then
      $F_i \leftarrow evaluate\_inner\_looop(G_{ij}, k_{in})$
    else
      $Ph_i \leftarrow G_i$ // Map to phenotype
      $F_i \leftarrow$ 0.5− expct-MCTS($Ph_i$, $depth$, $node$) // subtract from 0.5 as we normalise for zero sum games
return MAX($F$)
end function

function $evaluate\_inner\_looop(G_i, k_{in})$
$G_{initial} \leftarrow G_i$
Initialise Evolutionary Run
for l $\leftarrow$ 0 to $l = k_{in}$ do
  for each $G_{ij}$ in $G$ do
    $Ph_i \leftarrow G_i$ // Map to phenotype
    $Ph_{ij} \leftarrow G_{ij}$ // Map to phenotype
    $F_{ij} \leftarrow$ either play some games between the agents or evaluate exactly
return MAX($F$)
end function

function expct-MCTS($Ph_i$, $depth$, $node$)
if $node$.isTerminal() then
  return $\sum_{i=0}^{n} P(h_i)f(h_i, a)$
else if $depth < 0$ and $node$.isNotRandom() then
  return MCTS($node$, $NN_i$) {perform normal single player MCTS, with the same probabilities for each node as in the non-MCTS part (see below)}
else if $node$.isNotRandom() then
  $a \leftarrow -\infty$
  for each $child$ in $node$ do
    $a \leftarrow$ max(a, expct-MCTS($child$, $depth - 1$))
else
  if $node$.isNature() then
    for each $belief$ in $node$ do
      $P(h_i^{new}) := \sum_{i=0}^{k} P(h_i^{old})P(h_i^{old} \to h_i^{new})$ {Random Nature event. Initial beliefs are set to 1.0}
      expct-MCTS($child$, $depth - 1$)
  if $node$.isNatureFromOtherPlayers() then
    $a \leftarrow 0$
    for each $child$ in $node$ do
      for each $belief$ in $node$ do
        $P(h_i|a_k) := \frac{P(a_k|h_i)P(h_i)}{P(a_k)}$ {Bayesian belief update, using strategy $\sigma_i$, which is encoded in the neural network}
      $a \leftarrow a + P(O, H)$max(a, expct-MCTS($Ph_i$,$child$, $depth - 1$))
return $a$
end function
```

3.4 Tree Search solution

The problem with the previous solution is that it can take a long time to converge in larger problems. We need at least $|G|k_{in}k_{out}$ function evaluations for each generation, which is indeed slow, with $|G|$ being the number of players at each generation. An alternative would be to roll-out a tree and perform a tree search for the worst case scenario. However, it could be the case that not all actions are observable in our perspective environment, which makes things a bit harder. At this point we can start using Bayesian reasoning in order to form beliefs over the action space - which we no longer observe. In other words, we can replace the internal evolutionary run with a model-based tree search procedure, at least in the case where we can effectively discretize the action space and the states. Thus, we can use treat the "external" agent as nature. In that sense, we have created a POMDP; We then try to find the expected value for this POMDP by converting it to an MDP and performing Monte Carlo tree searches in order to get the expected value of the worst case opponent.

3.4.1 Converting the POMDP to an MDP

If the game is of perfect information, we can simply use the natural MDP that arises from the tree representation of the single player game. Assuming however that there are hidden actions, one can convert the POMDP to an MDP [15][3]. Hence, for each belief $h_i \in H$ (eq.7) and the latest action a_k:

$$P(h_i|a_k) := \frac{P(a_k|h_i)P(h_i)}{P(a_k)} \tag{7}$$

Thus, assuming we have a model for our opponents, $P(h_i)$ (the *prior*) becomes the probability of the hidden state in the current node (initially, for example, the probability of dealing card in kuhn's poker), before we make the current observation. $P(\alpha_k|h_i)$ (the *likelihood*) is the probability that one of our opponents would do an action, given that state h_i is true; this we obtain from our opponent model (if we include the histories/other observations as well). $P(\alpha_k)$ (the *marginal*) we get from the model again, by simply summing over all the probabilities for the last specific action in the tree.

Assuming now that nature performs a hidden action, and we are to move to a new set of states for each h_i, and with probability of moving from belief state h_i^{old} to h_i^{new} via nature hidden action a defined as $P(h_i^{old} \to h_i^{new})$, we can update our beliefs as follows (eq. 8):

$$P(h_i^{new}) := \sum_{i=0}^{k} P(h_i^{old})P(h_i^{old} \to h_i^{new}) \tag{8}$$

Once we reach the terminal nodes, we now have the final probability of each hypothesis. Thus the rewards are simply (eq. 9):

$$f(s, a) = \sum_{i=0}^{n} P(h_i)f(h_i, a) \tag{9}$$

Once this game is in this form, we can now use expectimax to find the reward that the best strategy would give us, the same way that we can do it in games of perfect information. Since expectimax's memory and speed requirements are huge, we only use expectimax to a certain depth, after which we use Monte Carlo Tree searches in order to acquire the average reward. The full algorithm is presented in Algorithm1.

[3]We convert to a belief MDP the same way as [15], albeit using a slightly different notation we think is more suitable for games

4. EXPERIMENTS

We have provided two sets of experiments; one where we just confirm that the algorithm works if it is used in games of known equilibria and another one where we try to evolve agents for full scale two-player limit texas holdem.

4.1 Simple Experiments

In the first batch of experiments, we don't use Monte Carlo tree searches, as the trees are small enough to be searched exhaustively. In the first experiment we use a vector representation of the actions (we are effectively evolving a population of agents with genome size three) using the purely evolutionary version of the algorithm presented above. We used a biased version of Rock-Paper-Scissors, first presented in [24]. The algorithm converges to the required equilibrium

- P(rock) = 0.0625
- P(paper) = 0.6250
- P(scissors) = 0.3125

with arbitrary precision within milliseconds. We can easily get similar results by evolving a neural network with 0 inputs[4] (none is needed, as there is only a single state), 1 hidden neuron and three output neurons (one for each probability of Rock, Paper or Scissor).

We can confirm that the algorithm does provide one of the solutions to Kunh's poker [16] $((1/3(0,0,3) + 1/3(0,1,3) + 1/3(2,0,3) = 1/3(00,00,11) + 1/3(00,01,11) + 1/3(10,00,11))$ (for player two), using the original paper's encoding. In our case the behavioural strategies found after 2000 iterations(using a neural network representation and full histories) are:

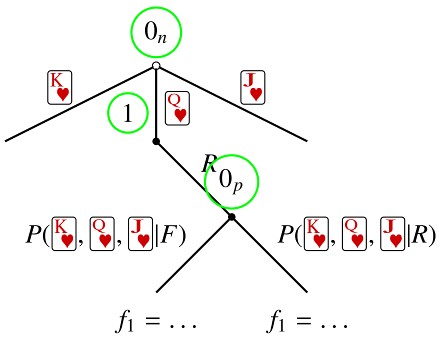


Figure 3: Part of Kuhn's poker tree

- $P(Bet|(1, OnBet)) = 0.001$
- $P(Bet|(2, OnBet)) = 0.333$
- $P(Bet|(3, OnBet)) = 0.999$
- $P(Bet|(1, OnCall)) = 0.333$
- $P(Bet|(2, OnCall)) = 0.001$
- $P(Bet|(3, OnCall)) = 0.999.$

We can also confirm that for one of the solutions for 13 card "one-card poker"[5], the tree search part of the algorithm measures the total maximum exploitation value as 5 log-precisions close to 0. An example tree for both Kuhn's poker and simple card poker can be seen in figure 3. In all the card games above we evolved a neural network with three inputs (current card, player one previous action, player two previous action), 4 hidden neurons and 2 outputs (one for raise/call and one for fold).

[4]A zero input neural network is the equivalent of a vector, we just demonstrate here that it can be evolved

[5]http://www.cs.cmu.edu/ ∼ggordon/poker/

4.2 Full Scale two player limit Holdem

We will not describe limit-holdem here, as we think there is an impressive amount of commercial publications tackling the subject. However, it is worth saying that holdem is a card game of poker, with multiple betting rounds. It is an imperfect game, with imperfection stemming from the fact that nature is performing hidden moves. It is also incomplete, due to the existence of nature (the card deck). The game, even in the 2 player abstracted version can have a huge number of tree nodes ($10^7 - 10^8$). In order to evolve full scale poker players for heads up limit holdem, we performed a number of heuristics so we could create a sensible abstraction. The resulting miximax tree is still huge (tens of millions of nodes), but it's manageable by today's computers. The process of abstraction followed is similiar to the one presented in [13]. First, we abstracted the game into 6 buckets using K-Means++ [1] with only one dimension. In order to extract features for each round, we map the expected value of each 2-card, 5-card, 6-card and 7-card combination in a probabilistic fashion, using Monte Carlo sampling. We perform $4,000,000$ for each of the 169 possible preflop hands, 100 iterations for each resulting sequence for flop and turn, while we perform 10 random rollouts for each river combination.We then associate each bucket with the probabilities of moving to another bucket in the next round (see figure 5).

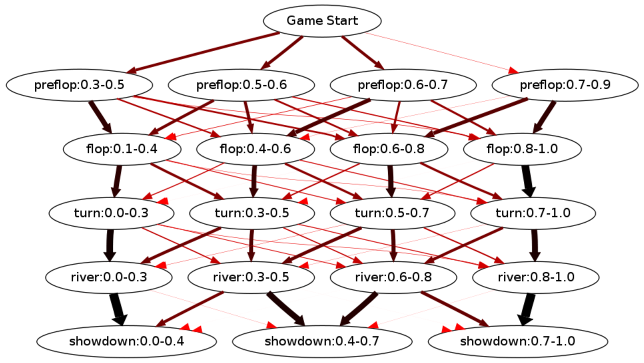
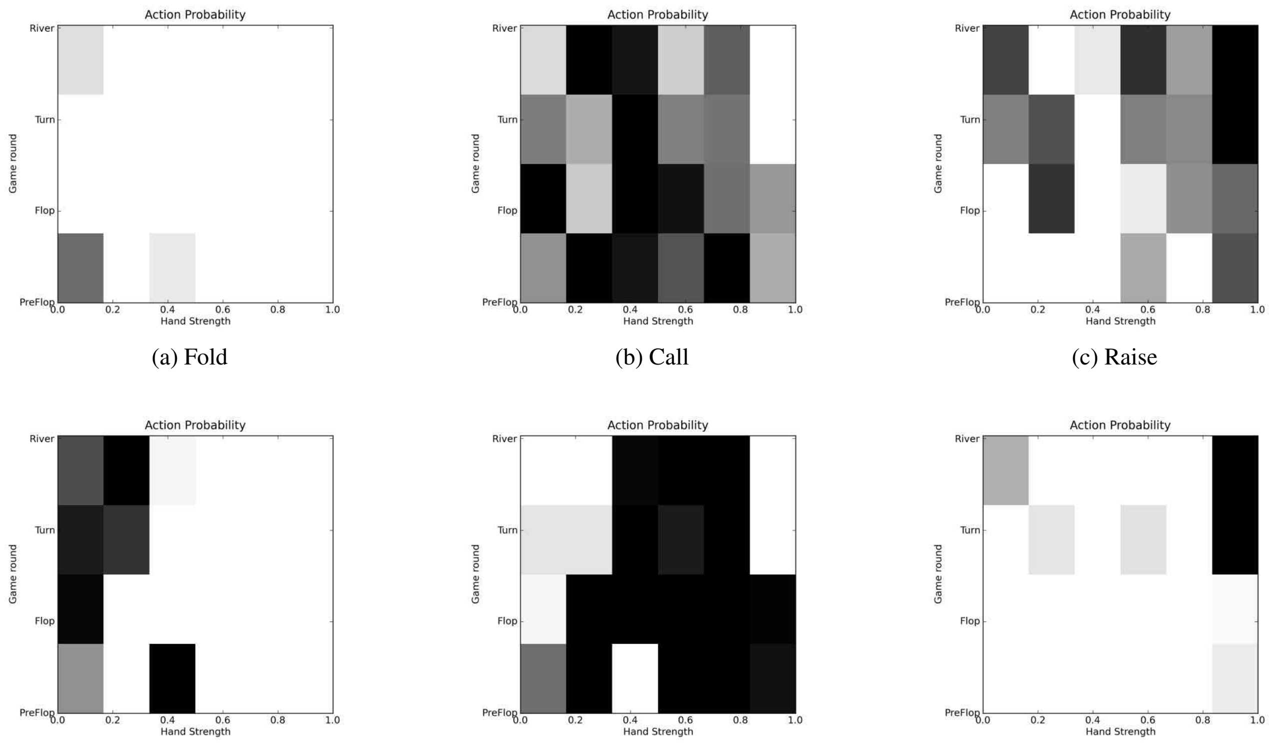


Figure 5: 4 Bucket clusters for each round of poker (for illustration purposes, we use 6-bucket clusters in our experiments). Please note that due to K-Means++ there are different probabilities for each each cluster, as it groups a different number of cards. Thicker lines represent higher transition probabilities, while the number of each circle represent the minimum and max expected value in a bucket.

For the agent, we evolve a set of tuples, each of which has three possible actions (fold, call, raise). Each tuple contains the following "inputs"

- The current bucket (i.e. from 0 to 5)
- The current round (from 0 to 3)
- The current position of the button (0 or 1)
- The raises of our current player in this round
- The opponent raises in this round.



(a) Fold (b) Call (c) Raise

(d) Fold (e) Call (f) Raise

Figure 4: Intensity plots for agent actions (no raises at all on top row, 1 raise in current round from the opponents on bottom row)

We evolved the agent for 200 generations, with a population of 16, which took approximately 12 hours[6]. The resulting agent is exploitable by 0.84 bets (with 1.5 bets being the the exploitability of an always folding player). The bets are set to 1 in the first two rounds (pre-flop and flop) and 2 during turn and river. For MCTS we used a maximum expectimax depth of 9, from which point onwards we used Monte Carlo tree searches for one thousand iterations. The progress can be seen in figure 6.

In figure 4 we can see an intensity plot for each action on each round, with darker colours meaning higher probability of an action. On the top row, we can see the agent's behaviour when there has been no raise. The same agent's behaviour can be observed in the bottom row, but this time when when a single raise took place during the current round. One can see that the agent is reasonable, although there are obviously still some inconsistencies (for example on the top row, an agent which hadn't yet seen any action might fold on preflop on some rare occasions, a senseless action).

We think its premature to compare directly with the best known algorithm to date for finding equilibria in large games, Counterfactual regret(CFR) [29]), as we have both imperfect recall and used significantly less computing power than CFR. We do require significantly less space for our policy (orders of magnitude) as we use a function approximator. However it is our belief that we can now attack larger games where the amount of computing resources required is currently prohibitive for CFR, albeit with less accuracy.

[6]We use a Core i7 950 machine in all experiments, with 8 parallel threads

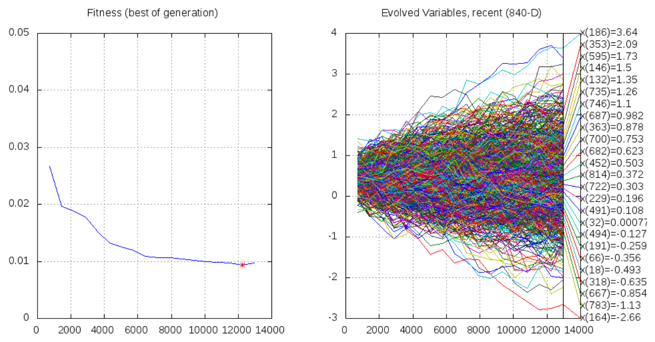


Figure 6: CMA-ES converging towards towards Nash VS number of function evaluations. The first graph on the left is the external fitness, while the graph on the right portrays the evolved genome

5. DISCUSSION & CONCLUSION

We have presented a novel method for approaching Nash equilibria in multi-player games. Two different algorithms were presented and evaluated, one based purely on evolutionary methods, and one mixing evolution with Monte Carlo tree searches. To the best of our knowledge this is the first time behavioural strategies have been directly attacked by coevolution (in the context of game-theoretic optimal solutions) and the first time one can approach Nash equilibria with arbitrary precision using evolutionary methods alone. Our method lacks convergence proofs, as it mostly depend on the quality of the genetic algorithm/evolutionary strategy used, and this might not be appealing to a number of game theory practitioners. We think however that especially in A.I., clean cut and dry solutions are not (or should not be) the aim. Our aim is to create methods that can potentially scale to massive problems, and we think we are on the right track with this one.

We understand that the second experiment required knowledge outside the coevolutionary domain, but there is literally no way of escaping the trivialities of most game theoretic problems without moving to large domains that require coarser abstractions. This requires bringing in multi-domain knowledge in order to even have a chance to attack the problem successfully.

Although the algorithm was created with games of imperfect information in mind, there is no reason why it cannot be used in games of complete/perfect information. We also think the algorithm can be used in order to find robust solutions in any problem where there is some form of dynamical change at runtime and we aim for robustness, as it directly attacks the minimax problem. There are two obvious next steps; the first one is to scale in unsolved games (ex. no limit holdem) and try to acquire strong agents in games where this couldn't be the case previously. The second one is to try and understand what kinds of agent representations are not only theoretically correct, but also produce strong agents in practice. For example, multi-layer perceptrons are notoriously hard to evolve for some games. It could be preferable that a different representation might be used, as to boost the performance of the final agents.

Acknowledgements

This research is partly funded by a postgraduate studentship from the Engineering and Physical Sciences Research Council.

6. REFERENCES

[1] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

[3] D. Billings, A. Davidson, T. Schauenberg, N. Burch, M. Bowling, R. Holte, J. Schaeffer, and D. Szafron. Game-tree search with adaptation in stochastic imperfect-information games. *Lecture Notes in Computer Science*, 3846:21–34, 2006.

[4] K. Binmore. *Playing for real: a text on game theory*. Oxford University Press, USA, 2007.

[5] D. Cliff and G. Miller. Co-evolution of pursuit and evasion II: Simulation methods and results. *From animals to animats*, 4:506–515, 1996.

[6] R. Dawkins and J. Krebs. Arms races between and within species. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, 205(1161):489–511, 1979.

[7] E. De Jong. Intransitivity in coevolution. In *Parallel Problem Solving from Nature-PPSN VIII*, pages 843–851. Springer, 2004.

[8] S. Ficici and J. Pollack. A game-theoretic approach to the simple coevolutionary algorithm. In *Parallel Problem Solving from Nature PPSN VI*, pages 467–476. Springer, 2000.

[9] S. G. Ficici. *Solution concepts in coevolutionary algorithms*. PhD thesis, Waltham, MA, USA, 2004.

[10] S. Gelly and Y. Wang. Exploration exploitation in go: UCT for Monte-Carlo go. In *Twentieth Annual Conference on Neural Information Processing Systems (NIPS 2006)*, 2006.

[11] N. Hansen, S. Muller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

[12] D. Harper. Competitive foraging in mallards. *Animal Behaviour*, 30(2):575–584, 1982.

[13] M. Johanson. Robust strategies and counter-strategies: Building a champion level computer poker player. Master's thesis, 2007.

[14] J. Kable and P. Glimcher. The neurobiology of decision: consensus and controversy. *Neuron*, 63(6):733–745, 2009.

[15] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[16] H. Kuhn. A simplified two-person poker. *Contributions to the Theory of Games*, 1:97–103, 1950.

[17] E. Manning. Using resource-limited nash memory to improve an othello evaluation function. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(1):40 –53, 2010.

[18] J. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America*, pages 48–49, 1950.

[19] S. Nolfi and D. Floreano. Coevolving Predator and Prey Robots: Do "Arms Races" Arise in Artificial Evolution? *Artificial Life*, 4(4):311–335, 1998.

[20] M. Osborne and A. Rubinstein. *A course in game theory*. The MIT press, 1994.

[21] J. Paredis. Coevolutionary computation. *Artificial Life*, 2(4):355–375, 1995.

[22] E. Popovici, A. Bucci, R. Wiegand, and E. de Jong. Coevolutionary Principles. *Handbook of Natural Computing*, 2010.

[23] S. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Prentice hall, 2009.

[24] M. Shafiei, N. Sturtevant, and J. Schaeffer. Comparing UCT versus CFR in Simultaneous Games. In *Proceedings of the IJCAI-09 Workshop on General Game Playing (GIGA'09)*, 2009.

[25] H. Siegelmann and E. Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4(6):77–80, 1991.

[26] L. Smith and M. Gasser. The development of embodied cognition: Six lessons from babies. *Artificial Life*, 11(1-2):30, 2005.

[27] R. Watson, J. Pollack, et al. Coevolutionary dynamics in a minimal substrate. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-01*, pages 702–709, 2001.

[28] W. Yoshida, R. Dolan, and K. Friston. Game theory of mind. *PLoS computational biology*, 4(12):1–14, 2008.

[29] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret minimization in games with incomplete information. *Advances in Neural Information Processing Systems*, 20:1729–1736, 2008.