The Parallel Nash Memory for Asymmetric Games

Frans A. Oliehoek
Informatics Institute,
University of Amsterdam
Kruislaan 403, 1098 SJ
Amsterdam, The Netherlands
faolieho@science.uva.nl

Edwin D. de Jong
Institute of Information and
Computing Sciences,
Utrecht University
PO Box 80.089, 3508 TB
Utrecht, The Netherlands
dejong@cs.uu.nl

Nikos Vlassis
Informatics Institute,
University of Amsterdam
Kruislaan 403, 1098 SJ
Amsterdam, The Netherlands
vlassis@science.uva.nl

ABSTRACT

Coevolutionary algorithms search for test cases as part of the search process. The resulting adaptive evaluation function takes away the need to define a fixed evaluation function, but may also be unstable and thereby prevent reliable progress. Recent work in coevolution has therefore focused on algorithms that guarantee progress with respect to a given solution concept. The Nash Memory archive guarantees monotonicity with respect to the game-theoretic solution concept of the Nash equilibrium, but is limited to symmetric games. We present an extension of the Nash Memory that guarantees monotonicity for asymmetric games. The Parallel Nash Memory is demonstrated in experiments, and its performance on general sum games is discussed.

Categories and Subject Descriptors

F.0 [**General**]

General Terms

Algorithms, Experimentation, Performance

Keywords

Coevolution, coevolution archive, Nash Memory, monotonic progress, game theory, Nash equilibrium

1. INTRODUCTION

Coevolution makes it possible to search for solutions to test-based problems without the need to specify a fixed evaluation function [1, 24, 16, 21, 23]. Rather than evaluating individuals with such a fixed fitness function, as done in most other approaches to evolutionary computation, coevolution simultaneously searches the space of tests that are used to evaluate the candidate solutions. These tests can in principle take any form, and test-based coevolution can be used to address a wide variety of problems that includes learning in games, concept learning, and function approximation.

The promise of coevolution follows from both theoretical considerations and practical demonstrations. Regarding the former, the biases that accompany a hand-designed fitness function can in principle be avoided, and the development of complex evaluation cases that would be difficult to construct by hand can in principle be performed by the search process itself. For instance, in order to evaluate the difference in strength between two high quality chess strategies, a sophisticated opponent strategy is required. In a non-coevolutionary approach, this opponent must somehow be available before the search process begins, which is problematic, as the goal of the search process itself was to identify a sophisticated game strategy; in coevolution, high-quality opponent strategies may be developed in the course of the search, when they are required.

Practical demonstrations of the potential of coevolution include the evolution of sorting networks [16], density classification using cellular automata [18, 12, 17, 30, 32], pursuit and evasion [37, 10, 4], function approximation and classification [29, 31] , the evolution of complex behavior [14, 42], and games such as backgammon [33], Tron [15], checkers, and Go [39] . While we focus on the application of coevolution to test-based problems here, coevolution can also be used to address problems where a fitness function is given; this form of coevolution is called *Cooperative* or *Compositional Coevolution* [35, 48, 49, 2].

While the potential of coevolutionary approaches is clear, the use of an adaptive evaluation function also introduces new problems. Unless carefully designed, the adaptive evaluation function can lead to a variety of problems that prevent stable progress [3, 47, 7]. A central current question in coevolution research therefore is how stable progress can be guaranteed.

As Ficici eloquently argues in his thesis [8], the question of how progress can be guaranteed must be preceded by a consideration of the desired *solution concept*. The solution concept specifies precisely which candidate solutions qualify as optimal solutions and which do not. Many, if not most disappointing results in earlier coevolution work can be traced back to incongruence between the employed algorithm and the desired solution concept. We briefly review the main solution concepts employed so far in test-based coevolution.

We follow the specification of solution concepts given in [6]. The simplest solution concept, $S0$, requires an opti-

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.

	R	P	S
R	0	−1	+1
P	+1	0	−1
S	−1	+1	0

A

	R	P	S
R	0	+1	−1
P	−1	0	+1
S	+1	−1	0

B

Table 1: The payoff matrices for Rock-scissors-paper. A: the payoff matrix of the row player. B: the column player's payoff matrix.

mal solution C to maximize the outcome over all possible tests simultaneously. The *covering competitive algorithm* [39] guarantees monotonicity for the $S0$ solution concept, and Schmitt [40] presents a convergence proof for a variant of $S0$ involving more than two types of individuals ('species'). For most problems of practical interest however, no single solution exists that simultaneously maximizes the outcomes of all tests, in which case the algorithms for this solution concept cannot be applied.

Solution concept $S1$, maximizing expected utility, specifies as a solution the individuals that maximize the expected score against a randomly selected opponent. Monotonic progress with respect to this solution concept is provided by the MaxSolve algorithm [6].

Pareto-coevolution [11, 46] views each test as a separate objective. The entire set of non-dominated solutions can be specified as a solution concept ($S3$). However, this set may contain many equivalent candidate solutions that each solve the same combination of tests. $S4$ is a variant of this solution concept that avoids such duplicate candidate solutions. Monotonic progress for $S4$ is guaranteed by the IPCA algorithm [5].

In this paper we adopt the solution concept of the Nash equilibrium ($S2$) for 2-player games [28]. In this setting, the set of individual solutions is the set of all pure strategies for player 1, whereas the set of tests is the set of all pure strategies for player 2. A Nash equilibrium specifies a pair of mixed (randomized) strategies, one for each player, such that no player has an incentive to unilaterally deviate given the strategies of the other players. From a game-theoretic perspective, a solution to a game is a recommendation for both players how to play. Clearly such a solution has to be a Nash equilibrium as otherwise at least one player would be better off using a different strategy (if it is not a Nash equilibrium, per definition there is a player who has an incentive to deviate). It is this same reasoning that also has led research in the field of multi-agent systems to focus on the Nash equilibrium as a solution concept [43]. For symmetric zero-sum games, Ficici's Nash Memory [8, 13] guarantees monotonicity for the solution concept of the Nash equilibrium, provided that no information is ever discarded [9].

The original description of the Nash Memory [13] was restricted to symmetric games (although it was already recognized that this could be extended to asymmetric games). In a 2-player symmetric game, e.g. Rock-Scissors-Paper which is shown in table 1, both players select their strategy from the same set of pure strategies available for the game, and the payoffs for the players are symmetric, i.e., the payoff matrix of the first player is the transpose of the second player's payoff matrix: $\mathbf{A} = \mathbf{B}^T$. If the game is zero-sum, additionally $\mathbf{A} = -\mathbf{B}$ holds. However, many problems of practical interest are asymmetric; examples of asymmetric games are backgammon, chess, poker, etc. In fact, *all* games in which players act in a turnwise fashion and observe (some of) each other's actions are asymmetric: a second player can always condition his action on the action of the first player in this case.

We present an extension of the Nash Memory to asymmetric games called the *Parallel Nash Memory*. This extension significantly widens the class of problems to which the Nash Memory algorithm can be applied. In particular, we address the case of finite *extensive form games* [22] or any game that can be cast as such. The Parallel Nash Memory is an archive method, and can be combined with any method for finding new strategies. Thus, any coevolutionary algorithm can be combined with the Parallel Nash Memory to produce a setup that guarantees monotonic progress for asymmetric games. In this paper, we focus on the behavior of the archive. Therefore, as a search heuristic we use a method that finds a best response strategy, based on solving a partially observable Markov Decision Process appropriately constructed from the corresponding finite extensive form game.

The structure of this paper is as follows. First, we describe extensive form games in Section 2. In Section 3, we describe the Nash Memory as introduced by Ficici. Section 4 presents the Parallel Nash Memory that will be investigated. Section 5 describes the best-response search heuristic employed in the experiments. The experiments are reported in Section 6. Section 7 discusses the applicability of to general sum games. Finally, Section 8 concludes.

2. EXTENSIVE FORM GAMES

Ficici's Nash Memory mechanism [13] deals with symmetric games in *normal-* or *strategic form*. We will now discuss a richer model that allows us to reason over a larger class of agent interactions, with several interesting applications. The model is a well known tree representation of agent dynamics originating from the classical game theory and called the *extensive form* representation. Here we will introduce this framework for partial information games.

An extensive form game [22] is given by a tree, the *game-tree*, in which nodes represent game states and whose root is the starting state. There are two types of non-terminal nodes: decision nodes that represent points at which agents can make a move, and chance nodes which represent stochastic transitions 'taken by nature'. Terminal nodes, or *outcome* nodes are the leaves of the game-tree. These specify the payoff for each agent.

In a partial information game, an agent may be uncertain about the true state of the game. This is reflected by the fact that an agent may not be able to discriminate between some nodes in the tree. Such groups of nodes in which the agent has the same information regarding the state are called *information sets*.

A *pure-* or *deterministic strategy* for an extensive form game is a mapping from information sets to actions. It is also possible to have strategies that allow randomization. There are two types of such strategies. A *mixed strategy* is a set of pure strategies together with a probability distribution over this set. A *stochastic strategy* is a mapping from information sets to probability distributions over actions. For more details we refer to [25].

3. THE NASH MEMORY MECHANISM

In this section, we describe the *Nash Memory* algorithm introduced by Ficici [13], and which forms the basis for the Parallel Nash Memory algorithm that will be presented here. The Nash Memory mechanism presents a method for iteratively reaching a Nash equilibrium for two-player symmetric zero-sum games.

In a two-player zero-sum game, a Nash equilibrium provides a security level payoff (the maximin value) for both players. Also, these maximin values sum to 0, so if the maximin value for the first player is v, that for the second player is $-v$. In symmetric zero-sum games, however, the expected payoff of a strategy played against itself is 0. Let E_1, E_2 denote the expected payoff for respectively the first and second player (we also write simply E for E_1). In symmetric zero-sum games holds $\forall_\pi\ E_1(\pi,\pi) = E_2(\pi,\pi) = 0$. As a consequence a Nash strategy should provide a security-level payoff of 0. Let $S(\pi)$ denote the *security set* of strategy π, i.e., $S(\pi) = \{\pi' | E(\pi,\pi') \geq 0\}$. We are thus searching for a strategy π, such that $\forall_{\pi'}\ \pi' \in S(\pi)$.

Let $\mathcal{N}$ and $\mathcal{M}$ be two mutually exclusive sets of pure strategies. $\mathcal{N}$ is defined to be the support of mixed strategy $\pi_\mathcal{N}$, which will be the approximation of the Nash strategy during the coevolution process. The set $\mathcal{M}$ will contain encountered strategies that are not in the support of $\pi_\mathcal{N}$. These strategies are not needed by $\pi_\mathcal{N}$ in order for the latter to be secure against all encountered strategies, i.e. $\mathcal{N} \cup \mathcal{M} \subseteq S(\pi_\mathcal{N})$. In contrast to [13], we will not put any bounds on the size of the memory $\mathcal{M}$, in order to guarantee convergence.

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.

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

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

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

Next, a payoff matrix $\mathbf{A}$ of all strategies in $\mathcal{M} \cup \mathcal{N} \cup \mathcal{W}$ played against each other is constructed. In this matrix, the rows represent the pure strategies of the first player and the columns those of the second player. Matrix entry (i, j) gives the payoff (for player 1) of strategy i played against strategy j. Together with the constraint that the probabilities of a mixed strategy must sum to 1 and the desire of both players to maximize their own payoff, $\mathbf{A}$ can be converted to a linear program (LP) [44], which then can be solved [38]. The result will be a new mixed strategy $\pi'_\mathcal{N}$, the strategies to which it assigns positive weight, $\mathcal{N}'$, and the other strategies, $\mathcal{M}'$. $\pi'_\mathcal{N}$ is a Nash equilibrium in the sub-game formed by restricting the full game to the encountered policies, meaning it provides a security level payoff of 0 within this sub-game.

[1] In [13] a different initialization is proposed: the initial set $\mathcal{N}$ is also provided by the search heuristic.

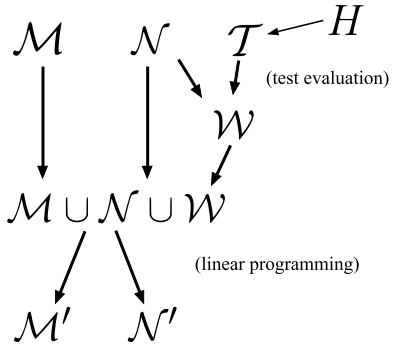

Figure 1: One iteration of the Nash Memory.

At this point a new iteration is started. The whole process is repeated until convergence.

This convergence is guaranteed: although $S(\pi'_\mathcal{N})$ is not necessarily a strict superset of $S(\pi_\mathcal{N})$[2], we know that $S(\pi'_\mathcal{N}) \supseteq \mathcal{M} \cup \mathcal{N} \cup \mathcal{W}$ holds for every iteration and that the set of encountered policies $\mathcal{M} \cup \mathcal{N} \cup \mathcal{W}$ grows monotonically. As the set of pure strategies is bounded (because we consider finite games), this means that in the worst case after a finite number of iterations the full game is constructed for which the Nash approximation is an actual Nash equilibrium. Therefore the algorithm has to converge to a Nash equilibrium.

Related is the definition of *monotonic solution concepts*, as presented in [9]. Let a sub-game formed by restricting a symmetric 2-player game to a set of pure policies $\mathcal{A}$ be denoted by $G_\mathcal{A}$. Similar $G_\mathcal{B}$, $G_\mathcal{C}$ denote sub-games restricted to sets $\mathcal{B}$ and $\mathcal{C}$. Loosely speaking, a monotonic solution concept means that the 'quality' of the solution found in each iteration monotonically improves as more strategies are encountered (provided that old strategies are never discarded). This 'quality' is defined with respect to a preference relation that specifies that a solution π_S is preferred to another π'_S, if all the games for which π'_S is the solution are proper subgames of games for which π_S is the solution. Formally, for monotonic solution concept S the following holds: If π_S is a (possibly mixed) strategy that is a solution for $G_\mathcal{A}$ and for $G_\mathcal{C}$, then

$$\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{C} \Rightarrow \pi_S \text{ is a solution for } G_\mathcal{B}$$

Ficici [9] proves that the Nash equilibrium is a monotonic concept for symmetric 2 player games: We know $\pi_S \in \mathcal{B}$ as $\mathcal{B} \supseteq \mathcal{A}$. If π_S is not a Nash equilibrium in $G_\mathcal{B}$, then $\mathcal{B}$ must contain a policy that is a better response to π_S than π_S itself, however as $\mathcal{B} \subseteq \mathcal{C}$, $\mathcal{C}$ should also contain this better response which conflicts with the premise that π_S is a Nash equilibrium for $G_\mathcal{C}$.

4. ASYMMETRIC NASH MEMORY

In the following, we present a generalization that renders the Nash Memory applicable to asymmetric zero-sum games. We will describe two methods: a naive method, and the proposed Parallel Nash Memory.

[2] $\pi'_\mathcal{N}$ might not attain positive payoff against a yet unencountered strategy against which $\pi_\mathcal{N}$ did attain positive payoff. Also, $S(\pi_\mathcal{N}) \not\supseteq S(\pi'_\mathcal{N})$ as $\pi'_\mathcal{N}$ is secure against the winners $\mathcal{W}$ of last iteration and $\pi_\mathcal{N}$, per definition, is not.

4.1 Naive symmetrization

A simple way to extend the Nash Memory mechanism to asymmetric 2-player games, is by converting asymmetric games to symmetric games. This can be done very easily by defining a new compound game consisting of two copies of the original asymmetric games: one played as the first player and one played as the second player. This compound game is symmetric and a particular strategy π^i is given by $\pi^i = \langle \pi_1^i, \pi_2^i \rangle$. I.e., the set of policies in the compound game is the Cartesian product of the set of player 1 and player 2 policies. The expected payoff of two policies $\pi^i = \langle \pi_1^i, \pi_2^i \rangle$, $\pi^j = \langle \pi_1^j, \pi_2^j \rangle$ for the compound game is given by the sum of the expected payoffs of their 'sub-policies' played against each other:

$$E(\pi^i, \pi^j) = E(\pi_1^i, \pi_2^j) + E(\pi_1^j, \pi_2^i)$$

Using this representation the Nash Memory mechanism can directly be applied without changes.

However, it is clear that the flexibility with which the new mixed strategy is constructed is constrained. Observe that it is not possible to put more weight on a particular first player strategy π_1^i without putting the same weight on the corresponding second player strategy π_2^i. For this reason we refer to this method as *naive symmetrization*.

4.2 Parallel Nash Memory

Here we will present another way of extending the Nash Memory to deal with asymmetric games. This approach is more sophisticated than naive symmetrization and referred to as the *parallel Nash Memory*. Note that in the Nash Memory's operation (section 3), there are only two reasons why the game must be symmetric: to determine whether a test strategy $T \in \mathcal{T}$ beats the current mixed strategy, $E(T, \pi_\mathcal{N}) > 0$, and because the next Nash approximation is calculated from one large set containing all encountered strategies (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 strategy approximation $\pi_{p,\mathcal{N}}$ for both players $p = 1, 2$. As mentioned in section 3, in zero-sum games if the Nash approximation is an actual Nash equilibrium, the strategies $\pi_{1,\mathcal{N}}$ and $\pi_{2,\mathcal{N}}$ should both provide a security level payoff (the maximin levels for the players), and these payoffs should sum to 0. We use this information to decide whether the Nash approximations should be updated when new strategies are found by the search heuristic. Now, if, without loss of generality, we assume that the search heuristic delivers a single test strategy for both players, T_1 and T_2, we can test whether the compound strategy $T = \langle T_2, T_1 \rangle$ [3] beats the compound strategy $\pi_\mathcal{N} = \langle \pi_{1,\mathcal{N}}, \pi_{2,\mathcal{N}} \rangle$, 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 strategy 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 encountered pure strategies for the respective players. By constructing the payoff matrix $\mathbf{A}$ for these pure strategies and applying linear programming we calculate the new Nash approximation $\pi'_\mathcal{N} = \langle \pi'_{1,\mathcal{N}}, \pi'_{2,\mathcal{N}} \rangle$, finishing one iteration.

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

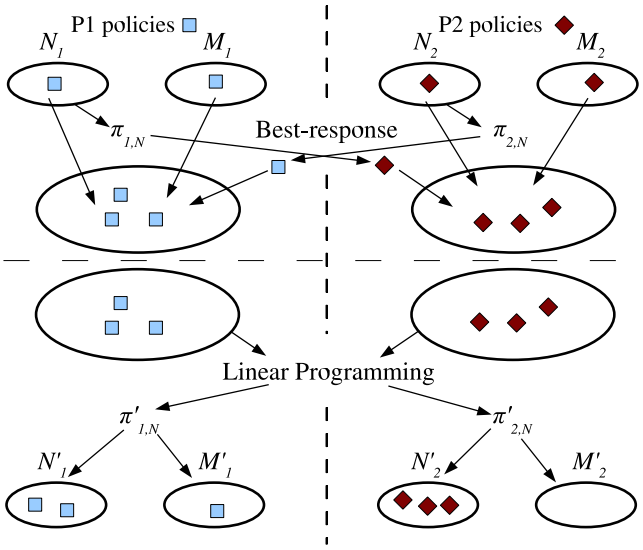

Figure 2: An iteration of parallel Nash Memory using the best-response heuristic.

In figure 2 one iteration (the 2nd) of the parallel Nash Memory is illustrated. In the first step T_1 and T_2 are calculated by using the best-response heuristic, i.e., by calculating a best-response against the current Nash policy approximations $\pi_{1,\mathcal{N}}$ and $\pi_{2,\mathcal{N}}$. Because of the use of the best-response heuristic, the test $E(T, \pi_\mathcal{N}) > 0$ is in this case unnecessary, as further discussed in the next section. In the second step a LP is constructed from all the encountered strategies and subsequently solved. This results in the new Nash approximation $\pi'_\mathcal{N} = \langle \pi'_{1,\mathcal{N}}, \pi'_{2,\mathcal{N}} \rangle$. As before, $\pi'_\mathcal{N}$ is an exact Nash equilibrium for the sub-game formed by restricting the full game to the encountered policies and therefore both $\pi'_{1,\mathcal{N}}$ and $\pi'_{2,\mathcal{N}}$ give the best worst-case payoff, i.e. the security level payoff for this sub-game. The new sets $\mathcal{N}'_1, \mathcal{N}'_2$ are constructed as the support of $\pi'_{1,\mathcal{N}}$ and $\pi'_{2,\mathcal{N}}$. The remaining policies are stored in $\mathcal{M}'_1$ and $\mathcal{M}'_2$.

Convergence is still guaranteed for the parallel Nash Memory. We know now that: $S(\pi'_{1,\mathcal{N}}) \supseteq \mathcal{M}_2 \cup \mathcal{N}_2 \cup \mathcal{W}_2$ and $S(\pi'_{2,\mathcal{N}}) \supseteq \mathcal{M}_1 \cup \mathcal{N}_1 \cup \mathcal{W}_1$ holds for every iteration and that the sets of encountered policies $\mathcal{N}_p$, $\mathcal{T}_p$, $\mathcal{W}_p$ for both players $p = 1, 2$ grow monotonically. Again, because the sets of pure strategies are bounded, in the worst case we will construct the full game for which the Nash approximation is an actual Nash equilibrium, so the algorithm has to converge to a Nash equilibrium.

Similarly, the Nash equilibrium is a monotonic solution concept for asymmetric games. Suppose that $\pi = \langle \pi_1, \pi_2 \rangle$ is a Nash equilibrium for $G_\mathcal{A}$ and $G_\mathcal{C}$, that $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{C}$, but π is not a Nash equilibrium for $G_\mathcal{B}$. This would mean that either player would have a better response policy in game $G_\mathcal{B}$. Suppose this is player 1, so this means that there is a $\pi'_1 \in \mathcal{B}$ that is a better response to π_2. However, as $\mathcal{B} \subseteq \mathcal{C}$ $\pi'_1 \in \mathcal{C}$, this conflicts with the premise that π is a Nash equilibrium for $G_\mathcal{C}$ and therefore can hold. The same argument holds for player 2.

5. BEST-RESPONSE SEARCH HEURISTIC

Here we discuss 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. In our setting this means that it has to find strategies that beat the current Nash approximation.

In principle, any coevolutionary algorithm can be used as a search heuristic. However, in order to test the behavior of the memory mechanism, rather than the search heuristic employed, we use a maximally effective search heuristic here that identifies a best response strategy to the current archive.

As illustrated in [26], when the stochastic strategy of the opponent is fixed and known (e.g. estimated from repeated play), an extensive form game can be recast as a partially observable Markov decision process (POMDP) [41] from the perspective of the protagonist agent. For more details on this transformation, we refer to [26, 27]. It is known that solving a POMDP gives an optimal deterministic strategy [36], which we will call a best-response strategy. This best-response strategy gives the highest payoff attainable against the fixed opponent strategy. Solving a POMDP exactly is intractable in general. However, in the special case of an extensive form game, 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 [36].

Calculating a best-response against the current Nash approximation is an effective search heuristic as this provides 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 strategies do not attain a positive payoff in the compound game, $E(T, \pi_{\mathcal{N}}) = E(T_2, \pi_{2,\mathcal{N}}) + E(T_1, \pi_{1,\mathcal{N}}) = 0$, then $\pi_{\mathcal{N}}$ is a Nash strategy and the algorithm converged.

There is still one problem to be overcome, however. We can calculate a best-response against a stochastic strategy [26]. In contrast, the Nash approximations, are mixed strategies, making it necessary to convert such a mixed strategy to a stochastic strategy. Proof that this is possible and the accompanying algorithm is given in [25]. 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 strategy μ. Every pure strategy 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 strategy, π, was responsible for realizing I. This is very closely related to the notion of 'realization weights' [19].

6. EXPERIMENTS

Here we describe some experiments we performed on a simple poker game from literature [20], called *8-card poker*. We also performed experiments on a similar, but larger poker variant.

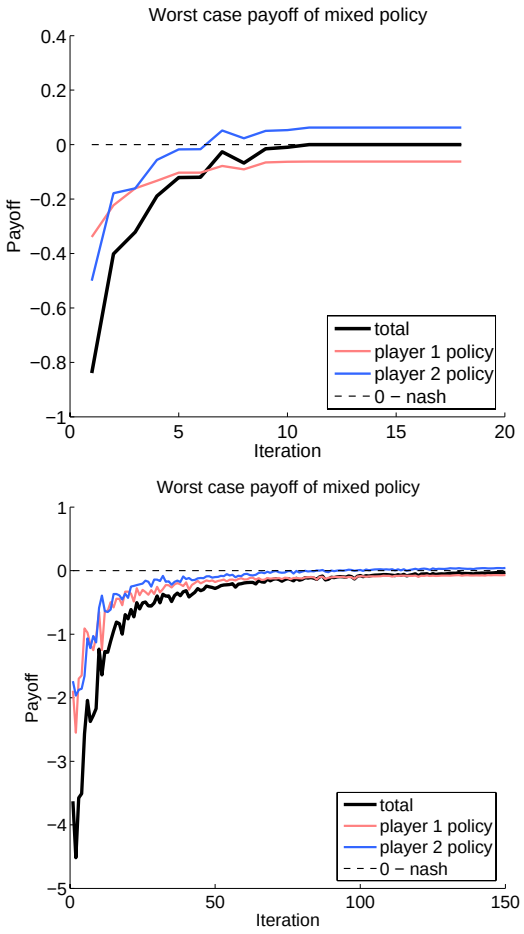

Figure 3: Results for the parallel Nash Memory approach. Top: 8-card poker. Bottom: 2-round, 3-bets-per-round 6-card poker.

6.1 8-card poker

This poker variant is a zero-sum game 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.

We implemented the parallel Nash Memory using best-response heuristic and applied it to 8-card poker. Figure 3 shows the results. It only takes a few iterations to obtain a strategy that is fairly secure, implying that this technique might be applied for larger games to obtain an approximate Nash strategy.

The figure also indicates that only a relatively small number of pure strategies are needed to make up a secure mixed

strategy. It turns out that the number of pure strategies used by the mixed strategy is even lower than the figure suggests: when reaching the Nash level (iteration 12) only 6 out of 12 pure strategies are assigned weight for both the gambler and the dealer strategy.

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

6.2 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 strategies needed for the support of the mixed strategy was also confirmed for the larger games: at iteration 150 for the 6-card poker game[4], the number of strategies with positive weight was 29 for both players.

7. NON ZERO-SUM GAMES

In this section, we discuss the applicability of the parallel Nash memory using a best-response heuristic to identical- and general payoff games. In games with identical payoff, the payoff matrix of the two players is equal, i.e.: $\mathbf{A} = \mathbf{B}$. In this case, linear programming doesn't apply anymore. Instead, the players select the pair of pure strategies that maximize the payoff. This is illustrated in table 2. Shown are two payoff matrices with arbitrarily picked numbers. Assume the Nash approximations $\pi^1_{1,\mathcal{N}}, \pi^1_{2,\mathcal{N}}$ in the first iteration are pure strategies. The best-responses to $\pi^1_{1,\mathcal{N}}, \pi^1_{2,\mathcal{N}}$ calculated in this first iteration are T^1_1, T^1_2.[5] The constructed payoff matrix $\mathbf{A}$ shows that the strategy pair $\langle T^1_2, \pi^1_{2,\mathcal{N}} \rangle$ yields the highest payoff for both players and consequently is the only Nash equilibrium in this sub-game. Therefore the players will select the pure strategies $\pi^2_{1,\mathcal{N}} = T^1_2$ and $\pi^2_{2,\mathcal{N}} = \pi^1_{2,\mathcal{N}}$ as their new Nash approximation for iteration 2.

In iteration 2, again two best-responses will be calculated. However, as $\pi^2_{2,\mathcal{N}} = \pi^1_{2,\mathcal{N}}$ the new best-response will be identical to the previous, i.e. $T^2_2 = T^1_2$, and therefore the expected payoff of $\langle T^2_2, \pi^2_{2,\mathcal{N}} \rangle$ will remain 3.4. This means that an improvement can only come from the new best-response T^2_1. Note that, because only the newly calculated pure strategies will be assigned weight as they are guaranteed to provide a better payoff for both players, it is not necessary to retain the previously found strategy in the memory $\mathcal{M}$. In this way, for identical payoff games, the parallel Nash Memory effectively reduces to *coordinate ascent* or *alternating maximization*.

	$\pi^1_{2,\mathcal{N}}$	T^1_1
$\pi^1_{1,\mathcal{N}}$	3	3.3
T^1_2	3.4	2.7

	$\pi^2_{2,\mathcal{N}}(=\pi^1_{2,\mathcal{N}})$	T^2_1
$\pi^2_{1,\mathcal{N}}(=T^1_2)$	3.4	3.6
$T^2_2(=T^1_2)$	3.4	3.6

Table 2: The identical payoff matrix A. Left: iteration 1. Right iteration 2.

In contrast to 2-player zero-sum games, where all Nash equilibria are guaranteed to have the same value, in identical payoff games there can be Nash equilibria specifying different values. Parallel Nash memory will converge to an arbitrary Nash equilibrium which does not have to be Pareto optimal. This corresponds to the fact that coordinate ascent methods are only guaranteed to find local optima.

A general payoff game, where $\mathbf{A}$ and $\mathbf{B}$ are two independent matrices, can be translated to a linearly complementary problem (LCP) and subsequently solved using for example the Lemke-Howson algorithm [19]. This suggests that we can adapt the Nash memory for general payoff games by replacing the LP by a LCP. Because for a general payoff game a Nash equilibrium is only guaranteed to exist in randomized strategies, like for zero-sum games, it will be necessary to retain previously encountered strategies in the memory $\mathcal{M}$, as solving the LCP will calculate a mixture over all encountered strategies. Because the Nash equilibria can specify different payoffs, like for identical payoff games, the result can only be guaranteed to be a locally optimal joint strategy. By enumerating all Nash equilibria in the sub-game formed by restriction to the encountered policies [19, 45], it is possible to find the optimal solution within the sub-game, but this still doesn't need to be optimal for the full game.

The original convergence argument was based on the notion of security sets. Although this notion is no longer useful in the setting of general sum games, the algorithm is still guaranteed to converge. This can be seen as follows: solving the LCP at an iteration t will give a Nash equilibrium for the sub-game induced by the $t+1$ strategies[6] $\mathcal{N}_p \cup \mathcal{M}_p \cup T_p$ for each player p found up to this iteration. If this Nash equilibrium for the induced game is not a Nash equilibrium for the full game, that means that (for at least one of the players) there exists at least one pure[7] strategy in the full game to which he has an incentive to deviate and this strategy, per definition, is a best-response. The best-response search heuristic will identify such a strategy and include it in the next iteration. As there are only a finite amount of pure strategies, parallel Nash Memory using best-response search heuristic will always converge to a Nash equilibrium. When a different search heuristic is used, convergence is still guaranteed, provided that the search heuristic is powerfull enough. More specifically, there should be a non-zero probability that the search heuristic returns the best-response that is not yet included.

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

[5]Again, T^1_2 denotes the best-response against the player 2 Nash approximation $\pi^1_{2,\mathcal{N}}$ and therefore is a pure strategy for player 1.

[6]One new strategy per iteration plus the initial strategy.

[7]A mixed strategy is only a best-response when all the pure strategies it assigns positive support to are also best-responses.

8. CONCLUSION AND DISCUSSION

We presented an extension of the Nash Memory mechanism, the Parallel Nash Memory, that enables calculation of Nash equilibria for asymmetric games. The Parallel Nash Memory was combined with a best-response search heuristic for extensive form games. Experiments with zero-sum poker games were presented, and the applicability of the method to identical payoff games and general payoff games was discussed.

When comparing the performance of the setup used in this paper against an approach solving the sequence form as in [20], there are some interesting differences. Overall, the setup employed in the experiments here uses more time compared to sequence-form solving. However, it spends its time differently: most time is spent in the search heuristic during, in particular in constructing and solving the POMDP models and determining outcomes between the encountered pure strategies. Far less time is spent by the linear programming method that forms part of the archive itself, 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. Thus, the Parallel Nash Memory that has been presented may provide an efficient archive method for asymmetric coevolutionary problems.

Regarding the search heuristic that was employed, we expect that a considerable speed-up can be obtained by streamlining the implementation of POMDP model construction and solving. Furthermore, approximate methods could be used for both solving the POMDP and evaluating the rewards to achieve a further speedup.

Another idea for future work 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 strategies could be done using any of the methods described in [34].

Acknowledgments

The first author is funded by the Interactive Collaborative Information Systems (ICIS) project, supported by Dutch Ministry of Economic Affairs, grant nr: BSIK03024.

9. REFERENCES

[1] R. Axelrod. The evolution of strategies in the iterated prisoner's dilemma. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, Research Notes in Artificial Intelligence, pages 32–41, London, 1987. Pitman Publishing.

[2] A. Bucci and J. B. Pollack. On identifying global optima in cooperative coevolution. In *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 1, pages 539–544, Washington DC, USA, 25-29 June 2005. ACM Press.

[3] D. Cliff and G. F. Miller. Tracking the Red Queen: Measurements of adaptive progress in co-evolutionary simulations. In F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, editors, *Proceedings of the Third European Conference on Artificial Life: Advances in Artificial Life*, volume 929 of *LNAI*, pages 200–218, Berlin, 1995. Springer.

[4] D. Cliff and G. F. Miller. Co-evolution of pursuit and evasion II: Simulation methods and results. In P. Maes, M. J. Mataric, J.-A. Meyer, J. B. Pollack, and S. W. Wilson, editors, *From Animals to Animats. Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior, SAB-96*, volume 4, pages 506–515, Cambridge, MA, 1996. The MIT Press.

[5] E. D. De Jong. The Incremental Pareto-Coevolution Archive. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-04*, pages 525–536, 2004.

[6] E. D. De Jong. The maxsolve algorithm for coevolution. In H.-G. Beyer, editor, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-05*, pages 483–489, 2005.

[7] E. D. De Jong and J. B. Pollack. Ideal evaluation from coevolution. *Evolutionary Computation*, 12(2):159–192, 2004.

[8] S. G. Ficici. *Solution Concepts in Coevolutionary Algorithms*. PhD thesis, Brandeis University, 2004.

[9] S. G. Ficici. Monotonic solution concepts in coevolution. In *Genetic and Evolutionary Computation – GECCO-2005*, pages 499–506, 2005.

[10] S. G. Ficici and J. B. Pollack. Coevolving communicative behavior in a linear pursuer-evader game. In R. Pfeifer, B. Blumberg, J.-A. Meyer, and S. Wilson, editors, *From Animals to Animats. Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior, SAB-98*, pages 263–269, Cambridge, MA, 1998. The MIT Press.

[11] S. G. Ficici and J. B. Pollack. A game-theoretic approach to the simple coevolutionary algorithm. In M. Schoenauer et al., editor, *Parallel Problem Solving from Nature, PPSN-VI*, volume 1917 of *LNCS*, Berlin, 2000. Springer.

[12] S. G. Ficici and J. B. Pollack. Pareto optimality in coevolutionary learning. In J. Kelemen, editor, *Sixth European Conference on Artificial Life*, Berlin, 2001. Springer.

[13] S. G. Ficici and J. B. Pollack. A game-theoretic memory mechanism for coevolution. In *Genetic and Evolutionary Computation – GECCO-2003*, pages 286–297, 2003.

[14] D. Floreano, S. Nolfi, and F. Mondada. Competitive coevolutionary robotics: From theory to practice. In R. Pfeifer, B. Blumberg, J.-A. Meyer, and S. Wilson, editors, *From Animals to Animats. Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior, SAB-98*, pages 512–524, Cambridge, MA, 1998. The MIT Press.

[15] P. Funes. *Evolution of Complexity in Real-World Domains*. PhD thesis, Brandeis University, Waltham, MA, 2001.

[16] D. W. Hillis. Co-evolving parasites improve simulated evolution in an optimization procedure. *Physica D*, 42:228–234, 1990.

[17] H. Juillé and J. B. Pollack. Coevolutionary learning: a case study. In *Proceedings of the 15th International Conference on Machine Learning*, pages 251–259, San Francisco, CA, 1998. Morgan Kaufmann.

[18] H. Juillé and J. B. Pollack. Coevolving the "ideal" trainer: Application to the discovery of cellular automata rules. In *Proceedings of the Third Annual Genetic Programming Conference*, Madison, Wisconsin, 1998.

[19] D. Koller, N. Megiddo, and B. von Stengel. Fast algorithms for finding randomized strategies in game trees. In *Proc. of the 26th ACM Symposium on Theory of Computing (STOC)*, pages 750–759, 1994.

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

[21] J. R. Koza. Genetic evolution and co-evolution of computer programs. In C. Langton, C. Taylor, J. Farmer, and S. Rasmussen, editors, *Artificial Life II*, volume X of *SFI Studies in the Sciences of Complexity*, pages 603–629. Addison-Wesley, Redwood City, CA, 1992.

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

[23] K. Lindgren. Evolutionary phenomena in simple dynamics. In C. Langton, C. Taylor, J. Farmer, and S. Rasmussen, editors, *Artificial Life II*, volume X of *SFI Studies in the Sciences of Complexity*, pages 295–312. Addison-Wesley, Redwood City, CA, 1992.

[24] J. Miller. The coevolution of automata in the repeated prisoner's dilemma. Santa Fe Institute working paper 89-003, 1989.

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

[26] F. A. Oliehoek, M. T. J. Spaan, and N. 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, Feb. 2005.

[27] F. A. Oliehoek, N. Vlassis, and E. D. de Jong. Coevolutionary Nash in poker games. In *BNAIC 2005: Proceedings of the 17th Belgian-Dutch Conference on Artificial Intelligence*, pages 188–193, Oct. 2005.

[28] M. J. Osborne and A. Rubinstein. *A course in game theory*. The MIT Press, Cambridge, MA, 1994.

[29] L. Pagie and P. Hogeweg. Evolutionary consequences of coevolving targets. *Evolutionary Computation*, 5(4):401–418, 1998.

[30] L. Pagie and P. Hogeweg. Information integration and Red Queen dynamics in coevolutionary optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation, CEC-00*, pages 1260–1267, Piscataway, NJ, 2000. IEEE Press.

[31] J. Paredis. Coevolutionary computation. *Artificial Life*, 2(4), 1996.

[32] J. Paredis. Coevolving cellular automata: Be aware of the Red Queen. In T. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms, ICGA-97*, San Francisco, CA, 1997. Morgan Kaufmann.

[33] J. B. Pollack and A. D. Blair. Co-evolution in the successful learning of backgammon strategy. *Machine Learning*, 32(1):225–240, 1998.

[34] R. Porter, E. Nudelman, and Y. Shoham. Simple search methods for finding a Nash equilibrium. *Games and Economic Behavior*, (to appear).

[35] M. A. Potter and K. A. De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.

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

[37] C. W. Reynolds. Competition, coevolution and the game of tag. In R. A. Brooks and P. Maes, editors, *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 59–69, Cambridge, MA, 1994. The MIT Press.

[38] R. T. Rockafellar. *Convex analysis*. Princeton, N.J., Princeton University Press, 1970.

[39] C. D. Rosin. *Coevolutionary Search among Adversaries*. PhD thesis, University of California, San Diego, CA, 1997.

[40] L. M. Schmitt. Theory of coevolutionary genetic algorithms. In M. Guo and L. T. Yang, editors, *Parallel and Distributed Processing and Applications, International Symposium, ISPA 2003*, pages 285–293, Berlin, 2003. Springer.

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

[42] K. O. Stanley and R. Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21:63–100, 2004.

[43] N. Vlassis. A concise introduction to multiagent systems and distributed AI. Informatics Institute, University of Amsterdam, Sept. 2003. http://www.science.uva.nl/~vlassis/cimasdai.

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

[45] B. Von Stengel. Computing equilibria for two-person games. In R. Aumann and S. Hart, editors, *Handbook of Game Theory with Economic Applications*, volume 3, chapter 45, pages 1723–1759. Elsevier, 2002. http://ideas.repec.org/h/eee/gamchp/3-45.html.

[46] R. A. Watson and J. B. Pollack. Symbiotic combination as an alternative to sexual recombination in genetic algorithms. In M. Schoenauer et al., editor, *Parallel Problem Solving from Nature, PPSN-VI*, volume 1917 of *LNCS*, Berlin, 2000. Springer.

[47] R. A. Watson and J. B. Pollack. Coevolutionary dynamics in a minimal substrate. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-01*, pages 702–709, San Francisco, CA, 2001. Morgan Kaufmann.

[48] R. A. Watson and J. B. Pollack. A computational model of symbiotic composition in evolutionary transitions. *Biosystems*, 69(2-3):187–209, May 2003. Special Issue on Evolvability, ed. Nehaniv.

[49] R. P. Wiegand. *An Analysis of Cooperative Coevolutionary Algorithms*. PhD thesis, George Mason University, Fairfax, Virginia, 2003.