

Theoretical and Experimental Analysis of an Evolutionary Social-Learning Game

Ryan Carr, Eric Raboin, Austin Parker, and Dana Nau
University of Maryland, College Park, MD 20742, USA

January 13, 2012

Abstract

An important way to learn new actions and behaviors is by observing others, and several evolutionary games have been developed to investigate what learning strategies work best and how they might have evolved. In this paper we present an extensive set of mathematical and simulation results for Cultaptation, which is one of the best-known such games.

We derive a formula for measuring a strategy's expected reproductive success, provide algorithms to compute near-best-response strategies and near-Nash equilibria, and provide techniques for efficient implementation of those algorithms. Our experimental studies provide strong evidence for the following hypotheses:

1. The best strategies for Cultaptation and similar games are likely to be conditional ones in which the choice of action at each round is conditioned on the agent's accumulated experience. Such strategies (or close approximations of them) can be computed by doing a lookahead search that predicts how each possible choice of action at the current round is likely to affect future performance.
2. Such strategies are likely to exploit most of the time, but will have ways of quickly detecting structural shocks, so that they can switch quickly to innovation in order to learn how to respond to such shocks. This conflicts with the conventional wisdom that successful social-learning strategies are characterized by a high frequency of innovation; and agrees with recent experiments by others on human subjects that also challenge the conventional wisdom.

Keywords: Evolutionary Games, Social Learning, Cultaptation

1 Introduction

An important way to learn new actions and behaviors is social learning, i.e., learning by observing others. Some social-learning theorists believe this is how most human behavior is learned [1], and it also is important for many other animal species [11, 39, 33]. Such learning usually involves evaluating the outcomes of others' actions, rather than indiscriminate copying of others' behavior [8, 22], but much is unknown about what learning strategies work best and how they might have evolved.

For example, it seems natural to assume that communication has evolved due to the inherent superiority of copying others' success rather than learning on one's own via trial-and-error innovation. However, there has also been substantial work questioning this intuition [4, 25, 2, 28, 12].

Several evolutionary games have been developed to investigate social learning [29, 20, 27, 5]. One of the best-known is Cultaptation, a multi-agent social-learning game developed by a consortium of European scientists [5] who sponsored an international tournament with a €10,000 prize.¹ The rules of Cultaptation are rather complicated (see Section 2), but can be summarized as follows:

¹NOTE: None of us is affiliated with the tournament or with the Cultaptation project.

- Each agent has three kinds of possible actions: innovation, observation, and exploitation. These are highly simplified analogs of the following real-world activities, respectively: spending time and resources to learn something new, learning something by communicating with another agent, and exploiting the learned knowledge.
- At each step of the game, each agent must choose one of the available actions. How an agent does this constitutes the agent’s “social learning strategy.”
- Each action provides an immediate numeric payoff and/or information about the payoffs of other actions at the current round of the game. This information is not necessarily correct in subsequent rounds because the actions’ payoffs may vary from one round to the next, and the way in which they may vary is unknown to the agents in the game.²
- Each agent has a fixed probability of dying at each round. At each round, each agent may also produce offspring, with a probability that depends on how this agent’s average per-round payoff compares to the average per-round payoffs of the other agents in the game.

A second Cultaptation tournament is scheduled to begin in February 2012. This tournament carries a €25,000 prize and introduces a few new concepts into the game, such as the ability for agents to improve actions they already know, and proximity-based observation. This paper does not deal with these additions, although we plan to address them in future work.

Our work has had two main objectives: (1) to study the nature of Cultaptation to see what types of strategies are effective; and (2) more generally, to develop ways of analyzing evolutionary environments with social learning. Our results include the following:

1. Analyzing strategies’ reproductive success (Section 6). Given a Cultaptation game G and a set \mathbf{S} of available strategies for G , we derive a formula for approximating (to within any $\epsilon > 0$) the expected per-round utility, $EPRU(s | G, \mathbf{S})$, of each strategy in \mathbf{S} .³ We show that a strategy with maximal expected per-round utility will have the highest expected frequency in the limit, independent of the initial strategy profile. These results provide a basis for evaluating highly complex strategies such as the ones described below.

Generalizability: These results can be generalized to other evolutionary games in which agents live more than one generation, with a fixed probability of death at each generation, and reproduction is done using the replicator dynamic.

2. Computing near-best-response strategies (Section 7). We provide a strategy-generation algorithm that, given a Cultaptation game G and a set of available strategies \mathbf{S} , can construct a strategy s_α that is within ϵ of the a response to \mathbf{S} .⁴

Generalizability: The strategy-generation algorithm performs a finite-horizon search, and is generalizable to other evolutionary games in which there is a fixed upper bound on per-round utility and a nonzero lower bound on the probability of death at each round.

²For our analyses, we assume the payoffs at each round are determined by an arbitrary function (which may be either deterministic or probabilistic), and we analyze how strategies perform given various possible characteristics of that function. In general, such characteristics would not be known to any Cultaptation agent—but our objective is to examine the properties of strategies in various versions of Cultaptation, not to develop a Cultaptation agent *per se*.

³Because of how death and mutation work in Cultaptation, it follows that $EPRU(s | G, \mathbf{S})$ is the same for every initial strategy profile S composed of strategies in \mathbf{S} . In particular, it is the same regardless of how many agents are using each strategy when the game begins.

⁴More precisely, s_α is an ϵ -best response to any initial strategy profile composed of strategies in \mathbf{S} .

3. Approximating symmetric Nash equilibria (Section 8). We provide CSLA, an iterative self-improvement algorithm that uses the strategy-generation algorithm in Section 7 to produce a strategy s_{self} that is a near-best response in a Cultaptation game in which the other players are all using s_{self} . Hence a strategy profile composed entirely of instances of s_{self} is a symmetric near-Nash equilibrium.

Generalizability: An iterative self-improvement algorithm similar to CSLA should be able to approximate a Nash equilibrium for any game in which the strategies are complex enough that computing a best (or near-best) response is not feasible by analyzing the strategies directly, but is feasible using information from a simulated game between strategies in the profile. Games of this type will typically have a high branching factor but relatively simple interactions between agents.

4. State aggregation (Section 7.5). To make our algorithms fast enough for practical experimentation, we provide a state-aggregation technique that speeds them up by an exponential factor without any loss in accuracy. Our experimental results in Section 9 demonstrate the practical feasibility that this provides: in our experiments, CSLA always converged in just a few iterations.

Generalizability: The state-aggregation technique is generalizable to other evolutionary games in which the utilities are Markovian.

5. Experimental results (Section 9). In our experimental studies, the near-Nash equilibria produced by CSLA in any given game were all virtually identical, regardless of the starting values that we used. That strongly suggests (though it does not prove) that the strategy profile consisting of copies of s_{self} approximates an optimal Nash equilibrium, and possibly even a unique Nash equilibrium.

Consequently, s_{self} 's characteristics provide insights into the characteristics of good Cultaptation strategies. For example, our experiments show that s_{self} exploits most of the time, but switches quickly to innovation when a structural shock occurs, switching back to exploitation once it has learned how to respond to the shock. This conflicts with the conventional wisdom [35, 4] that successful social-learning strategies are characterized by a high frequency of innovation, but it helps to explain both the results of the Cultaptation tournament [34] and some recent experimental results on human subjects [38].

6. Implications. Our results provide strong support for the following hypotheses about the best strategies for Cultaptation and similar games:

- *What they are like, and how they can be computed.* The best strategies are likely to be conditional ones in which the choice of action at each round is conditioned on the agent's accumulated experience. Such strategies (or close approximations of them) can be computed by doing a lookahead search that predicts how each possible choice of action at the current round is likely to affect future performance.
- *How they are likely to behave.* It is likely that the best strategies will exploit most of the time, but will have ways of quickly detecting structural shocks, so that they can switch quickly to innovation in order to learn how to respond to such shocks.

2 Cultaptation Social-Learning Game

This section gives a more detailed description of the Cultaptation social learning game, adapted from the official description [5]. The game is a multi-agent round-based game, where one action is chosen by each agent each round. There are N agents playing the game, where N is a parameter to the game. No agent knows of any other agent's actions at any point in the game except through the Obs action specified below. The actions available to each agent are innovation (InV), observation (Obs), and exploitation (X_1, \dots, X_μ ,

where μ is a parameter to the game). Each **InV** and **Obs** action informs the agent what the utility would be for one of the exploitation actions, and an agent may not use an exploitation action X_i unless the agent has previously learned of it through an innovation or observation action. Here are some details:

Exploitation. Each exploitation action X_i provides utility specific to that action (e.g. X_1 may provide utility 10 and X_2 may provide utility 50). The utility assigned to each action at the beginning of the game is drawn from a probability distribution π , where π is a parameter to the game.

The utility provided by each exploitation action X_i may change on round r , according to a probability c_r . The function c is a parameter to the game, and specifies the *probability of change* for every round of the game. When the changes occur, they are invisible to the agents playing the game until the agent interacts with the changed action. For instance: if an action’s utility happens to change on the same round it is exploited, the agent receives the new utility, and discovers the change when the new utility is received. The new utility for a changed action is determined via the distribution π .

Innovation. When an agent uses the **InV** action, it provides no utility, but it tells the agent the name and utility of some exploitation action X_i that is chosen uniformly at random from the set of all exploitation actions about which the agent has no information. If an agent already knows all of the exploitation actions, then **InV** is illegal, and indeed undesirable (when there is nothing left to innovate, why innovate?). The agent receives no utility on any round where she chooses an **InV** action.

Observation. By performing an **Obs** action, an agent gets to observe the action performed and utility received by some other agent who performed an exploitation action on the previous round. Agents receive no utility for **Obs** actions, nor any information other than the action observed and its value: the agent being observed, for instance, is unknown. If none of the other agents performed an exploitation action on the previous round, then there were no X_i actions to observe so the observing agent receives no information. In some variants of the social learning game, agents receive information about more than one action when observing. We do not treat such variants directly in this paper, but it is straightforward to extend our algorithms to take this difference into account.

Example 1 Consider two strategies: the innovate-once strategy (hereafter **I1**), which innovates exactly once and exploits that innovated action (whatever it is) for the rest of the game, and the innovate-twice-observe-once strategy (hereafter **I2O**), which innovates twice, observes once, and exploits the highest valued action of the actions discovered for the rest of the game. For simplicity of exposition, suppose there are only four exploitation actions: X_1 , X_2 , X_3 , and X_4 . The values for each of these actions are drawn from a distribution; in this example we will assume that they are chosen to be 3, 5, 8, and 5, respectively. For simplicity, we will assume the probability of change is 0. Suppose there are two agents: one **I1** and one **I2O**. For the first action, **I1** will innovate, which we suppose gives **I1** the value of action X_1 . On every sequential action, **I1** will choose action X_1 , exploiting the initial investment. If the agent dies k rounds later, then the history of actions and utilities will be that given in Table 1; giving a utility of $3(k - 1)$ and a per-round utility of $3\frac{k-1}{k}$.

In contrast, **I2O** will innovate, informing it of the utility of X_3 : 8, then it will innovate again, informing it of the utility of X_4 : 5, and finally it will observe. On the second round, **I1** performed X_1 , and since these are the only two agents, this was the only exploitation action performed. Therefore, **I2O**’s observation action on the next round must report that another agent got a utility of 3 from action X_1 last round (if there were multiple possibilities, one would be chosen uniformly at random). On round 4, **I2O** then knows that actions X_1 , X_3 , and X_4 have utilities

Round #	1	2	3	4	5	...	k
I1's action	Inv	X_1	X_1	X_1	X_1	...	X_1
I1's utility	0	3	6	9	12	...	$3(k-1)$
Per round	0	1.5	2	2.25	2.4	...	$3\frac{k-1}{k}$
I2O's action	Inv	Inv	Obs	X_3	X_3	...	X_3
I2O's utility	0	0	0	8	16	...	$8(k-3)$
Per round	0	0	0	2	3.2	...	$8\frac{k-3}{k}$

Table 1: Action sequences from Example 1, and their utilities.

of 3, 8, and 5, respectively. Since the probability of change is 0, the obvious best action is X_3 , which I2O performs for the rest of her life. The utility of I2O on round k is $8(k-3)$, making the per-round utility $8\frac{k-3}{k}$. Note that on rounds 2 to 4, I2O will have a worse per-round utility than I1, while after round 4, the utility of I2O will be higher (this is important because reproduction is tied to per-round utility, as we will show shortly).

Formally, everything that an α knows about each round can be described by an *action-percept pair*, $(a, (m, v))$, where $a \in \{\text{Inv}, \text{Obs}, X_1, \dots, X_\mu\}$ is the action that α chose to perform, and (m, v) is the percept returned by the action. More specifically, $m \in \{X_1, \dots, X_\mu, \emptyset\}$ is either an exploitation action or a null value, and v is the utility observed or received. While a is chosen by the agent, m and v are percepts the agent receives in response to that choice. If a is **Inv** or **Obs**, then v is the utility of exploitation action m . If a is **Obs** and no agent performed an exploitation action last round, then there is no exploitation action to be observed, hence $m = \emptyset$ and $v = 0$. If a is some X_i , then m will be the same X_i and v will be the utility the agent receives for that action. The *agent history* for agent α is a sequence of such action-percept pairs, $h_\alpha = \langle (a_1, (m_1, v_1)), \dots, (a_k, (m_k, v_k)) \rangle$. As a special case, the empty (initial) history is $\langle \rangle$.

Example 2 The history for I2O in Example 1 is:

$$h_{I2O} = \langle (\text{Inv}, (X_3, 8)), (\text{Inv}, (X_4, 5)), (\text{Obs}, (X_1, 3)), (X_3, (X_3, 8)), \dots \rangle$$

To concatenate a new action-percept pair onto the end of a history, we use the \circ symbol. For example, $h_\alpha \circ (a, (m, v))$ is the history h_α concatenated with the action-percept pair $(a, (m, v))$. Further, for $h_\alpha = \langle p_1, p_2, \dots, p_k \rangle$, where each p_i is some action-percept pair, we let $h_\alpha[i] = p_i$, and $h_\alpha[i, \dots, j]$ be the subhistory $\langle p_i, \dots, p_j \rangle$.

Strategies. The Cultaptation game is ultimately a competition among *strategies*. Here, a strategy is a function from histories to the set of possible actions: $s : h_\alpha \mapsto m$, where h_α is a history of an agent using s and m is **Inv**, **Obs** or X_i for some i . Since each strategy may depend on the entire history, the set of possible strategies is huge;⁵ but any particular Cultaptation game is a competition among a much smaller set of strategies \mathbf{S} , which we will call the *set of available strategies*. For example, if there are n contestants, each of whom chooses a strategy to enter into the game, then in this case,

$$\mathbf{S} = \{\text{the strategies chosen by the contestants}\}. \quad (1)$$

⁵The number of possible mixed strategies is, of course, infinite. But even if we consider only pure strategies, the number is quite huge. We show in Appendix C that for a 10,000-round Cultaptation game of the type used in the Cultaptation tournament [34], a loose lower bound on the number of pure strategies is $100^{9.4 \times 10^{20155}}$. In contrast, it has been estimated [37] that the total number of atoms in the observable universe is only about 10^{78} to 10^{82} .

Each strategy in \mathbf{S} may be used by many different agents, and the strategy profile at each round of the game may change many times as the game progresses. When an agent reproduces, it passes its strategy on to a newly created agent, with the per-round utility of each agent determining its likelihood of reproduction. A strategy’s success is measured by its average prevalence over the last quarter of the game [5].

The replication dynamics work as follows. On each round, each agent has a 2% chance of dying. As such, we also include a parameter d in our formulation representing the probability of death (d defaults to 0.02). Upon death, an agent is removed from the game and replaced by a new agent, whose strategy is chosen using the *reproduction* and *mutation* mechanisms described below. Mutation happens 2% of the time, and reproduction happens 98% of the time.

Reproduction. When reproduction occurs, the social learning strategy used by the newborn agent is chosen from the strategies of agents currently alive with a probability proportional to their per-round utility (the utility gained by an agent divided by the number of rounds the agent has lived). The agent with the highest per-round utility is thus the most likely to propagate its strategy on reproduction. We now give an example of this.

Example 3 Again looking at the sequences of actions in Table 1, we see that both agents would have equal chance of reproducing on round 1. However, on round 2 I1 has a per-round utility of 1.5, while I2O has a per-round utility of 0, meaning I1 gets 100% of the reproductions occurring on round 2. Round three is the same, but on round 4 I1 has a per round utility of 2.25 and I2O has a per-round utility of 2. This means that I1 gets $100 \cdot 2.25/4.25 = 53\%$ of the reproductions and I2O gets $100 \cdot 2/4.25 = 47\%$ of the reproductions on round 4.

Mutation. In Cultaptation, *mutation* does not refer to changes in an agent’s codebase (as in genetic programming). Instead, it means that the new agent’s strategy s is chosen uniformly at random from the set of available strategies, regardless of whether any agents used s on the previous round. For instance, if there were a cultaptation game pitting strategies I1 and I2O against one another, then a new mutated agent would be equally likely to have either strategy I1 or I2O, even if there were no living agents with strategy I1.

Game Types. In the Cultaptation tournament [34], two types of games were played: pairwise games and melee games. A *pairwise* game was played with an invading strategy and a defending strategy. The defending strategy began play with a population of 100 agents, while the invading strategy began with none. Mutation was also disabled for the first 100 rounds, to allow the defending strategy time to begin earning utility. After 100 rounds, mutation was enabled and the invader had the challenging task of establishing a foothold in a population consisting entirely of agents using the defending strategy (most of whom would have had time to find several high-payoff actions). Since the pairwise games provide a clear early-game advantage to the defender, they were typically played twice with the invader and defender swapping roles on the second game. A *melee* game was played with n strategies, for some $n > 2$. Initially, the population of 100 agents was evenly divided between each strategy in the game. Mutation was disabled for the last quarter of the game, so that it would not influence results when strategies had similar fitness.

Scoring. If we have k social learning strategies s_1, \dots, s_k playing Cultaptation, then on any given round there will be some number n_j of agents using strategy s_j , for $1 \leq j \leq k$. Strategy s_j ’s *score* for the game is *the average value of n_j over the final 2,500 rounds of the game*. The strategy with the highest score is declared the winner.

The only way an agent may affect n_j is through reproduction. We will show in Section 6.2 that any strategy maximizing an agent’s expected per-round utility (defined in Section 5.5) will also maximize its reproduction. We will therefore focus on computing the expected per-round utility.

Round	X_1	X_2	X_3	X_4	A1	A2	A3
1	2	4	1	9	N/A	N/A	(Obs, (\emptyset , \cdot))
2	2	4	1	9	N/A	N/A	(Inv, (X_4 , 9))
3	2	4	1	9	Birth	N/A	(X_4 , (\cdot , 9))
4	5	4	1	9	(Obs, (X_4 , 9))	Birth	(X_4 , (\cdot , 9))
5	5	2	1	9	(X_4 , (\cdot , 9))	(Obs, (X_4 , 9))	(X_4 , (\cdot , 9))
6	1	2	1	9	(X_4 , (\cdot , 9))	(X_4 , (\cdot , 9))	(X_4 , (\cdot , 9))
7	1	2	8	9	(X_4 , (\cdot , 9))	(X_4 , (\cdot , 9))	Death
8	1	2	8	1	(X_4 , (\cdot , 1))	(X_4 , (\cdot , 1))	N/A
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots

Figure 1: An example of a game in which there is a large structural shock. The columns for the exploitation actions X_i show their values at each round, and the columns for agents A1–A3 show their histories. Note that by round 6, all agents choose action X_4 , which has changed to a very low value. Since none of the agents are innovating, none of them can find the newly optimal action X_3 .

3 Motivating Discussion

The purpose of this section is to explain the motivations for several aspects of our work:

- Sections 3.1 and 3.2 give examples of types of strategies that seem like they should work well at first glance, but can have unexpectedly bad consequences. The existence of such situations motivate the algorithms described later in this paper, which perform a game tree search in order to consider strategies’ long-term consequences.
- An important way of getting insight into a game is to examine its best-response strategies; and this approach is at the heart of our formal analysis and our game-tree search algorithms. Section 3.3 explains some issues that are important for finding best-response strategies in Cultaptation.

3.1 Innovation, Observation, and Structural Shocks

If we want to acquire a new action to exploit, then what is the best way of doing it: to observe, or to innovate? At first glance, the observing might seem to be the best approach. If the other agents in the environment are competent, then it is likely that they are exploiting actions that have high payoffs, hence we should be able to acquire a better action by observing them than by innovating. This suggests that an optimal agent will rely heavily on observation actions. However, the following example shows that relying *only* on observation actions can lead to disastrous consequences if there is a structural shock, i.e., a large change in the value of an exploitation action.⁶

Example 4 (structural shocks) Figure 1 shows a Cultaptation game in which all agents use the following strategy: each agent begins with a single Obs action, followed by a single Inv action if the Obs action returns \emptyset ,⁷ in order to obtain an exploitation action X_i which the agent will use in all subsequent rounds.

Agent A3 acquires action X_4 by doing an unsuccessful Obs followed by an Inv; and A1 and A2 acquire X_4 by observing A3. At first, X_4 is far better than the other exploitation actions, so all of the agents do well by using it. On round 8, the action X_4 changes to the lowest possible

⁶We have borrowed this term from the Economics literature, where it has an analogous meaning (e.g., [10, 14]).

⁷This will generally only happen on the first round of the game, before any agent has obtained an exploitation action.

value, but the agents continue to use it anyway. Furthermore, any time a new agent is born, it will observe them using X_4 and will start using it too.

This is a pathological case where the best action has disappeared and the agents are in a sense “stuck” exploiting the suboptimal result. Their only way out is if all agents die at once, so that one of the newly born agents is forced to innovate. In Section 9.1.2, our experiments show that in some cases, situations like these are a big enough risk that a near-best response strategy will choose innovation moves more frequently than observation moves.

3.2 Innovation and Observation Versus Exploitation

One might also think that agents should perform all of their innovation and observation actions first, so that they have as many options as possible when choosing an action to exploit. However, as Raboin et al. [32] demonstrate, this intuition is not always correct. Because the game selects which agents reproduce based on average per-round utility, not total accumulated utility, it is frequently better for newborn agents to exploit one of the first actions it encounters, even if this action has a mediocre payoff (e.g., exploiting an action with value 10 on the second round of an agent’s life gives it as much per-round payoff as exploiting an action with value 50 on the tenth round). Once the agent has at least some per-round utility so that it has a nonzero chance of reproducing, it can then begin searching for a high-valued action to exploit for the rest of its lifetime.

3.3 Best-Response Strategies in Cultaptation

A widely used technique for getting insight about a game (e.g., see [26]) is to look at the game’s best-response strategies. Given an agent α and a *strategy profile* (i.e., an assignment of strategies to agents) $\mathbf{s}_{-\alpha}$ for the agents other than α , α ’s *best response* is a strategy s_{opt} that maximizes α ’s expected utility if the other agents use the strategies in $\mathbf{s}_{-\alpha}$.

In Cultaptation, it is more useful to consider a best response to the set of available strategies \mathbf{S} , rather than any particular strategy profile. During the course of a Cultaptation game, the strategy profile will change many times as agents die and other agents are born to take their places. Each strategy in \mathbf{S} will be scored based on its average performance over the course of the game; and we can show (see Section 6.2.1) that given \mathbf{S} , each strategy’s score is independent of the initial strategy profile if the game is sufficiently long.

If G is a Cultaptation game (i.e., a set of values for game parameters such as the number of agents, set of available actions, probability distribution over their payoffs; see Section 5 for details), then for any agent α , any set of available strategies \mathbf{S} , and any history h_α for α , there is a probability distribution $\pi_{\text{Obs}}(a|h_\alpha, \mathbf{S})$ that gives the probability of observing each action a , given \mathbf{S} and h_α . Given π_{Obs} and G , we can calculate the probability of each possible outcome for each action our agent might take, which will allow us to determine the best response to \mathbf{S} . To compute π_{Obs} is not feasible except in general, but it is possible to compute approximations of it in some special cases (e.g., cases in which all of the agents, or all of the agents other than α , use the same strategy). That is the approach used in this paper.

4 Related Work

In this section we will discuss related work on social learning and on computational techniques related to our own.

4.1 Social Learning

The Cultaptation social learning competition offers insight into open questions in behavioral and cultural evolution. An analysis of the competition is provided by Rendell et al. [34]. Of the strategies entered into the competition, those that performed the best were those that greatly favored observation actions over innovation actions, and the top performing strategy learned almost exclusively through observation. This was considered surprising, since several strong arguments have previously been made for why social learning isn't purely beneficial [4, 35]. However, this result is consistent with observations made during our own experiments, in which the ϵ -best-response strategy rarely did innovation (see Section 9).

In previous work, Carr et al. showed how to compute optimal strategies for a highly simplified versions of the Cultaptation social learning game [6]. Their paper simplifies the game by completely removing the observation action—which prevents the agents from interacting with each other in any way whatsoever, thereby transforming the game into a single-agent game rather than a multi-agent game. Their model also assumes that exploitable actions cannot change value once they have been learned, which overlooks a key part of the full social learning game.

Wisdom and Goldstone attempted to study social learning strategies using a game similar to Cultaptation, but using humans rather than computer agents [38]. Their game environment consisted of a group of “creatures,” each of which had some hidden utility. The agents’ objective was to select a subset of the creatures to create a “team,” which was assigned a utility based on the creatures used to create it. Agents had a series of rounds in which to modify their team, and on each round they were allowed to see the teams chosen by other agents on the previous round (and in some cases, the utility of the other agents’ teams), and the object of the game was to maximize the utility of one’s team. In this game, the acts of keeping a creature on one’s team, choosing a creature that another agent has used, and choosing a creature no one has yet used correspond to exploitation, observation, and innovation (respectively) in the Cultaptation game.

The successful strategies Wisdom and Goldstone saw are similar to those used by the strategies found by our algorithm: they keep most of the creatures on their team the same from round to round (which corresponds in Cultaptation to performing mostly exploitation actions), and new creatures are mostly drawn from other agents’ teams (which corresponds to preferring observation over innovation in Cultaptation). However, Wisdom and Goldstone highlight these characteristics as interesting because they run contrary to the conventional wisdom for social learning strategies, which suggests that broader exploration should lead to better performance, and therefore that successful strategies should innovate more often [35]. In this case, analyzing the strategies found by our algorithm allowed us to draw the same conclusions about what works well. This gives more evidence that the conventional wisdom on social learning [4, 35] may be mistaken.

How best to learn in a social environment is still considered a nontrivial problem. Barnard and Sibly show that if a large portion of the population is learning only socially, and there are few information producers, then the utility of social learning goes down [2]. Thus, indiscriminate observation is not always the best strategy, and there are indeed situations where innovation is appropriate. Authors such as Laland have attempted to produce simple models for determining when one choice is preferable to the other [25]. Game theoretic approaches have also been used to explore this subject, but it is still ongoing research [15, 9]. Giraldeau et al. offer reasons why social information can become unreliable. Both biological factors, and the limitations of observation, can significantly degrade the quality of information learned socially [12].

Work by Nettle outlines the circumstances in which verbal communication is evolutionarily adaptive, and why few species have developed the ability to use language despite its apparent advantages [28]. Nettle uses a significantly simpler model than the Cultaptation game, but provides insight that may be useful to understanding social learning in general. In Nettle’s model, the population reaches an equilibrium at a point where both individual and social learning occur. The point of equilibrium is affected by the quality of observed information and the rate of change of the environment.

N	Number of agents in the environment.
\mathbf{S}	The set of available strategies. Agents may only use strategies in \mathbf{S} .
r	Number of the current round, ranging from 1 to ∞ .
c	The probability of change on all rounds.
d	The probability of death on all rounds.
$(a, (m, v))$	An action-percept pair in which the action a returns the percept (m, v) .
h_α	Agent history for α . A sequence of action-percept pairs experienced by agent α .
$h_\alpha[i]$	The i -th action-percept pair in h_α .
$X(h_\alpha)$	Number of exploitable actions given history h_α .
μ	Number of exploitation actions in the game.
π	Probability distribution for the new value of any action whose value changed at round r .
$\pi_{\text{Obs}}(m, v h_\alpha, \mathbf{S})$	Probability that Obs will observe action m with value v at history h_α .
$\pi_{\text{Inv}}(v r)$	Probability that Inv will return an action with value v on round r .
V	The set of potential utility values.
$P(h'_\alpha h_\alpha, a, \mathbf{S})$	Probability of transitioning to history h'_α if α performs action a at history h_α .
$L(h_\alpha)$	Probability that α lives long enough to experience history h_α .
T	Set of all action-percept pairs of the form $(a, (m, v))$.

Table 2: A glossary of notation used in this paper.

4.2 Related Computational Techniques

The *restless bandit* problem, a generalization of the stochastic multi-armed bandit problem that accounts for probability of change, is cited as the basis for the rules of the Cultaptation tournament [34]. The rules of the Cultaptation game differ from the *restless bandit* problem by including other agents, making observation actions possible and complicating the game significantly. We also show in Section 6.2 that maximizing total payoff, the goal of the restless bandit problem, is different from maximizing expected per-round utility (EPRU) of an agent in the Cultaptation tournament.

The *restless bandit* problem is known to be *PSPACE*-complete, meaning it is difficult to compute optimal solutions for in practice [30, 13]. Multi-armed bandit problems have previously been used to study the tradeoff between exploitation and exploration in learning environments [36, 24].

As discussed later in Section 5, finding a best-response strategy in Cultaptation is basically equivalent to finding an optimal policy for a Markov Decision Process. Consequently, our algorithm for finding near-best-response strategies has several similarities to the approach used by Kearns *et al.* to find near-optimal policies for large MDPs [21]. Both algorithms use the discount factor of the MDP (which, in our case, is the probability of death d) and the desired accuracy ϵ to create a horizon for their search, and the depth h_α of this horizon depends on the discount factor and the branching factor, but not on the size of the full state space (unlike conventional MDP algorithms). Thus, both their algorithm and ours also have running time exponential in $1/\epsilon$ and in the branching factor. However, the algorithm provided by Kearns *et al.* was designed as an online algorithm, so it only returns the near-optimal action for the state at the root of the search tree. Ours, on the other hand, returns a strategy specifying which action the agent should take for all states that can occur on the first h_α rounds. This means that our exponential-time algorithm only needs to run once to generate an entire strategy, rather than once per agent per round in each game we simulate.

Many algorithms for optimal control of an MDP have been developed, however they all have running time that grows linearly with the size of the state space of the MDP. This makes them intractable for problems like ours, which have exponentially large state spaces. Several approaches for *near-optimal* control, which produces a policy within some ϵ of optimal, have been developed [21, 23, 3].

5 Formal Model

In this section we introduce a formal mathematical model of Cultaptation games. A glossary of the notation used in this paper is provided as Table 2.

Game Definition. Cultaptation requires a number of parameters to determine exactly how it will run. Therefore, in our formal model, we will define the *game parameters*, G , to be a set of values for the following: N , the number of agents; μ , the number of exploitation actions in the game; c , the probability that an exploitation action changes its utility each round; π , the probability distribution used to assign a utility value to each exploitation action, both the outset and each time an action’s utility changes; and d , the probability of death. In the Cultaptation tournament, only the values of N , μ , and d were known ahead of time, but for our analysis we use the values of the other parameters as well.

Recall that in the Cultaptation tournament [5], each evolutionary simulation was a contest between two or more strategies submitted to the tournament. Thus, there is a fixed set of strategies that are allowed to occur in a given simulation. We will call this the *set of available strategies* \mathbf{S} , where $\mathbf{S} = \{s_1, s_2, \dots, s_\ell\}$ for some finite ℓ (i.e. in pairwise games $\ell = 2$, in melee games $\ell > 2$). Any strategy profile \mathbf{s} that occurs in the simulation will consist only of strategies in \mathbf{S} . When an agent is chosen to be replaced via mutation, its new strategy is selected at random from the strategies in \mathbf{S} .

We can now define a Cultaptation game formally, as follows. A *Cultaptation game* is an ℓ -player game, in which each player receives the game parameters G as input. Each player then simultaneously chooses a strategy to put into the set of available strategies. We will call player i ’s strategy s_i , so that $\mathbf{S} = \{s_1, s_2, \dots, s_\ell\}$. The pair (G, \mathbf{S}) is an *instance* of G . In (G, \mathbf{S}) , each player i will receive a payoff equal to $\text{score}(s_i)$, defined below.

Scoring. The version of Cultaptation used in the tournament continued for 10,000 rounds, and each strategy was assigned a score equal to its *average population* over the last 2,500 rounds. But as is often done in analyses of repeated games, our formal model assumes an *infinite Cultaptation game*, i.e., the game continues for an infinite number of rounds, and the score for strategy s is its average population over the entire game:

$$\text{score}(s) = \lim_{r \rightarrow \infty} \frac{\sum_{j=1}^r p(s, j)}{r},$$

where $p(s, j)$ is the population size of agents using strategy s on round j . This greatly simplifies our analysis in Section 5.5, by allowing us to average out the various sources of noise present in the game.

Actions. The rest of the formal model will be constructed from the perspective of an arbitrary agent, α , in a given infinite Cultaptation game instance (G, \mathbf{S}) . We use r for the number of a round, and $X(h_\alpha)$ to specify the number of exploitation actions available after history h_α . After all exploitation actions X_1, \dots, X_μ have been innovated or observed in a history h_α , then $X(h_\alpha) = \mu$ and innovation actions become illegal.

We model the payoffs supplied for exploitation actions X_i by a probability distribution π . $\pi(v)$ is the probability of an action having payoff v at the start of the game instance. $\pi(v)$ is also the probability that, when an action changes its payoff, the new payoff is v . We let V be the set of all action values that may occur with non-zero probability:

$$V = \{v \mid \pi(v) > 0\}.$$

We require the set V to be finite.

If we let $\pi_{\text{Inv}}(v|r)$ be the probability that value v is innovated on round r , it can be defined recursively in terms of c and π as:

$$\pi_{\text{Inv}}(v|r) = \begin{cases} \pi(v), & \text{if } r=0, \\ c\pi(v) + (1-c)\pi_{\text{Inv}}(v|r-1), & \text{otherwise.} \end{cases}$$

That is, initially the chance that `Inv` will return an action with value v is determined by the given distribution $\pi(v)$. On later rounds ($r > 0$) the chance that `Inv` will return an action with value v is the chance that an action's value changed to v on the current round (given by $c\pi(v)$), plus the chance that an action's value was v on the previous round and it did not change this round.

While computing the probability distribution for utilities of actions returned by `Inv` was fairly straightforward, computing a similar distribution for `Obs` actions is significantly more difficult. Let α be any agent, and \mathbf{S} be the set of available strategies. From \mathbf{S} we can get a probability distribution over the other agents' actions in any given situation; and from this we can derive $\pi_{\text{Obs}}(m, v|h_\alpha, \mathbf{S})$, the probability that `Obs` would return the action-percept pair (m, v) , given history h_α .

In order to derive π_{Obs} , we must consider each possible strategy profile $\mathbf{s}_{-\alpha}$ for agents besides α , determine how likely that strategy profile is to occur, and then determine what each agent in $\mathbf{s}_{-\alpha}$ will do for every possible sequence of actions they could have encountered, bounded only by the percepts our agent has received in h_α . As we discussed in Section 3.3, the number of possible histories alone is astronomically large. Since π_{Obs} is conditioned on each possible history it will be larger still, so in any practical implementation the best we can do is to approximate π_{Obs} (Section 7.5 describes how we will do this). But for our theoretical development, we will assume we have an oracle for π_{Obs} , that will tell us exactly how likely we are to observe any given action-utility pair.

In what follows, we will show that, given π , π_{Obs} , V , and \mathbf{S} , we can calculate the possible outcomes of each action the agent may take, and the probability of each of these outcomes. This allows us to treat an infinite Cultaptation game as a Markov Decision Process (MDP) [18]. Calculating the best response in this case is equivalent to finding an optimal control policy for an MDP.

5.1 Transition Probabilities

A transition probability function $P(h'_\alpha|h_\alpha, a, \mathbf{S})$ defines the probability of transitioning from history h_α to history $h'_\alpha = h_\alpha \circ (a, (m, v))$ in the next round if an agent α performs action a . These transition probabilities are for the case where α does not die before reaching h'_α ; we introduce functions to account for the probability of death in Section 5.2.

There are three cases for what $P(h'_\alpha|h_\alpha, a, \mathbf{S})$ might be, depending on whether a is an innovation, observation, or exploitation action:

- If $a = \text{Inv}$, then

$$P(h_\alpha \circ (\text{Inv}, (m, v))|h_\alpha, \text{Inv}, \mathbf{S}) = \begin{cases} \frac{\pi_{\text{Inv}}(v|r)}{\mu - X(h_\alpha)} & \text{if } h_\alpha \text{ contains no percepts that contain the action } m, \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

- Recall that an agent cannot innovate action m if it has already encountered m by innovating or observing. Observation actions are not subject to the same restriction, so if $a = \text{Obs}$, then

$$P(h_\alpha \circ (\text{Obs}, (m, v))|h_\alpha, \text{Obs}, \mathbf{S}) = \pi_{\text{Obs}}(m, v|h_\alpha, \mathbf{S}) \quad (3)$$

where $\pi_{\text{Obs}}(m, v|h_\alpha, \mathbf{S})$ models the exploitation behavior of the other agents in the environment. Obviously, the exact probability distribution will depend on the composition of strategies used by these agents. The above definition is general enough to support a wide range of environments; and in

Section 7.5 we will discuss one potential way to model this function for a more specific set of environments.

- Finally, if $a = X_m$, then h_α must contain at least one percept for X_m . Let r be the last round at which the last such percept occurred. For the case where X_m 's utility did not change since round r , we have

$$P(h_\alpha \circ (X_m, (m, v)) | h_\alpha, X_m, \mathbf{S}) = \underbrace{(1-c)^{|h_\alpha|-r}}_{\text{prob. of not changing}} + c\pi(v) \underbrace{\sum_{j=r}^{|h_\alpha|} (1-c)^{|h_\alpha|-j}}_{\text{prob. of changing back to } v} \quad (4)$$

For the case where X_m 's utility did change since round r , we have

$$P(h_\alpha \circ (X_m, (m, v)) | h_\alpha, X_m, \mathbf{S}) = c\pi(v) \sum_{j=r}^{|h_\alpha|} (1-c)^{|h_\alpha|-j} \quad (5)$$

which is similar, but assumes that the value must have changed at least once.

In all other cases, no transition from h_α to h'_α is possible, so $P(h'_\alpha | h_\alpha, a, \mathbf{S}) = 0$.

5.1.1 Probability of Reaching a History

We will frequently be interested in $P(h_\alpha | s, \mathbf{S})$, the probability of history h_α occurring given that the agent is following some strategy $s \in \mathbf{S}$. We will be able to derive $P(h_\alpha | s, \mathbf{S})$ iteratively, calculating the probability of each step of history h_α occurring using the functions derived above.

Specifically, $P(h_\alpha | s, \mathbf{S})$ is the probability that each $h_\alpha[i] = (a_i, (m_i, v_i))$ occurs given the action chosen by the strategy in the history $h_\alpha[1, \dots, i-1] = (a_1, (m_1, v_1)) \cdots (a_{i-1}, (m_{i-1}, v_{i-1}))$, or:

$$P(h_\alpha | s, \mathbf{S}) = \prod_{i=1}^{|h_\alpha|-1} P(h_\alpha[1, \dots, i] \circ h_\alpha[i+1] | h_\alpha[1, \dots, i], s(h_\alpha[1, \dots, i]), \mathbf{S}) \quad (6)$$

5.2 Accounting for Probability of Death

The probability of an agent living long enough to experience history h_α depends on the probability of death. It is

$$L(|h_\alpha|) = (1-d)^{|h_\alpha|-1}. \quad (7)$$

When we calculate the probability of reaching a given history h_α , we will generally multiply it by $L(|h_\alpha|)$ to account for the chance that the agent dies before reaching h_α .

Sometimes we will also be interested in the probability that a randomly-selected agent has history h_α . For this we will need to know the probability that a randomly-selected agent is exactly $|h_\alpha|$ rounds old, which is simply:

$$\frac{L(|h_\alpha|)}{\sum_{i=1}^{\infty} L(i)} = \frac{L(|h_\alpha|)}{\frac{1}{1-(1-d)}} = \frac{L(|h_\alpha|)}{\frac{1}{d}} = d L(|h_\alpha|). \quad (8)$$

5.3 Utility Functions

A utility function $U((a, (m, v)))$ defines the utility gleaned on action-percept pair $(a, (m, v))$:

$$U((a, (m, v))) = \begin{cases} v, & \text{if } \exists i \text{ such that } a = X_i, \\ 0, & \text{otherwise.} \end{cases} \quad (9)$$

Notice that $U(\cdot)$ is only non-zero on exploitation actions.

The *per-round utility (PRU)* of a history $h_\alpha = (a_1, (m_1, v_1)) \circ \dots \circ (a_{|h_\alpha|}, (m_{|h_\alpha|}, v_{|h_\alpha|}))$ is defined to be the sum of the utility acquired in that history divided by the history's length:

$$\text{PRU}(h_\alpha) = \frac{1}{|h_\alpha|} \sum_{i=1}^{|h_\alpha|} U((a_i, (m_i, v_i))) \quad (10)$$

5.4 Strategy Representation

A strategy s is defined as a function mapping each history $h_\alpha \in H$ to the agent's next action $s(h_\alpha) \in \{\text{Inv}, \text{Obs}, X_1, \dots, X_\mu\}$. For instance, the strategy s_{I1} from Example 1 is defined by the function:

$$s_{\text{I1}}(h_\alpha) = \begin{cases} \text{Inv}, & \text{if } h_\alpha \text{ is empty,} \\ X_i, & \text{for } h_\alpha = (\text{Inv}, (X_i, v)), \dots \end{cases}$$

In this paper we will deal with *partially specified* strategies. A partially specified strategy is a mixed strategy (i.e., a probability distribution over a set of pure strategies) that is defined by a finite set Q of history action pairs ($Q \subset H \times \{\text{Inv}, \text{Obs}, X_1, \dots, X_\mu\}$), in which each $h_\alpha \in H$ appears at most once. Given any history h_α , if there is an action m such that $(h_\alpha, m) \in Q$, then s_Q chooses the action m . Otherwise, s_Q chooses an action arbitrarily from all actions that are legal in h_α . Partially specified strategies have the advantage of being guaranteed to be finitely representable.

5.5 Evaluating Strategies

At each round, an agent with history h_α has reproductive fitness $\text{PRU}(h_\alpha)$, and agents are selected to reproduce with probability proportional to their reproductive fitness (i.e. using the replicator equation [16]). Since a strategy's score is a function of its average population over the course of the game, we want some metric that allows us to compare the expected reproductive fitness of two strategies. This will allow us to predict which strategy is more likely to win.

At first glance, it may appear that the way to predict which strategy will have higher expected reproductive fitness is to compare their expected utilities. However, prior work has shown that this is not the case: *in Cultaptation, a strategy's expected reproductive fitness is not necessarily proportional to its expected utility* [32]. We now present a simple example that illustrates this phenomenon.

Example 5 (reproductive fitness not proportional to expected utility) Consider an infinite Cultaptation game with no probability of change, no observation actions, probability of death $d = 0.05$, two exploitation actions valued at 65 and 100, and an innovate action that will return either exploitation action with uniform probability. This means that an agent needs to perform at most two innovate actions to have knowledge of the action with value 100, since innovating does not return an action the agent already knows.

We will compare two strategies: s_{II} and s_{IEI} . Both strategies will perform an innovate as their first action. If the action they learn has value 100, both strategies will exploit that action until the agent dies. If the action learned has value 65, s_{II} will perform a second innovate on its

Table 3: The expected utility and expected reproductive fitness of s_{II} and s_{IEI} from Example 5. s_{IEI} has a higher expected reproductive fitness, and therefore will be likely to win a game against s_{II} , even though s_{II} has a higher expected utility.

	Expected Utility	Expected Reproductive Fitness
s_{II}	90.25	65.074
s_{IEI}	88.825	65.185

next turn, learning the action with value 100, and will exploit that action until its agent dies. Meanwhile, s_{IEI} will exploit the action with value 65 once, before performing an innovate on its third turn to learn the action with value 100. It then exploits this action until its agent dies.

Since the two strategies are identical when they learn the action with value 100 on their first action, and since this case is equally likely to be encountered by both strategies, we can ignore it for the purposes of comparing them. For the rest of this analysis we will assume the first innovate returns the action with value 65. In this case, we can calculate the expected utility for both strategies using geometric series, and we can calculate their expected reproductive fitnesses using methods described in Section 6. Table 3 presents these values. While s_{II} has a higher expected utility, since it exploits the action with value 100 more often, s_{IEI} has a higher expected reproductive fitness, since it does not wait as long to begin exploiting. Therefore, s_{IEI} will be the likely winner in a contest between these two strategies.

Since we cannot always use a strategy’s expected utility to determine whether it is expected to win, we will instead compute a strategy’s expected reproductive fitness directly, by computing its Expected Per-Round Utility.

Definition. The *Expected Per-Round Utility* for a strategy s_α , $EPRU(s_\alpha | G, \mathbf{S})$, is the expected value of $PRU(h_\alpha)$ over all possible histories h_α for a randomly-selected agent α using strategy $s_\alpha \in \mathbf{S}$ in an infinite Cultaptation game instance (G, \mathbf{S}) . \square

To calculate $EPRU(s_\alpha | G, \mathbf{S})$, we look at each possible history h_α and multiply $PRU(h_\alpha)$ by the probability that a randomly-chosen agent using s_α has history h_α . This probability is equal to the probability that a randomly-chosen agent is $|h_\alpha|$ rounds old (Equation 8) times the probability of reaching history h_α in $|h_\alpha|$ steps using strategy s_α (Equation 6). Hence, the EPRU of a strategy is:

$$EPRU(s_\alpha | G, \mathbf{S}) = \sum_{h_\alpha \in H} \underbrace{d L(|h_\alpha|)}_{\text{Portion of agents } |h_\alpha| \text{ rounds old}} \times \underbrace{P(h_\alpha | s_\alpha, \mathbf{S})}_{\text{Chance of reaching } h_\alpha \text{ using } s} \times \underbrace{PRU(h_\alpha)}_{\text{Per-round utility}} .$$

Note that for a given environment, the probability of death d is a constant. Hence, in our analysis we will frequently factor it out.

Example 6 Recall the innovate-once strategy, which innovates once to learn an action and then exploits that action until it dies. Suppose this strategy exists in an environment with a probability of death of 0.2 and only one possible exploit action with non-changing value 10. All agents using this strategy will therefore learn the only action on their first round, and then exploit an action with value 10 on all subsequent rounds. Hence, there is only one possible history for a j -round old agent using this strategy, and its per-round utility is $10 \cdot (j-1)/j$. The probability that a randomly-selected agent will be j rounds old will be $0.2 \cdot L(j) = 0.2 \cdot 0.8^{j-1}$. Thus the expected per-round utility achieved by this strategy in this environment is $\sum_{j=1}^{\infty} 0.2 \cdot 0.8^{j-1} \cdot 10 \cdot (j-1)/j$.

6 Analysis of EPRU

In this section we examine methods for computing the expected per-round utility of a strategy. First we present a method for computing an approximation to the EPRU for given a strategy, then we present a proof that a strategy maximizing EPRU will also maximize its average population in an infinite Cultaptation game instance.

6.1 Computation of EPRU

We will now define a formula that can be used to compute EPRU exactly for a given strategy s . The definition of EPRU given in Section 5.5 used a “backward” view: for every possible history h_α , it looked back through h_α to determine $\text{PRU}(h_\alpha)$. Notice, however, that h_α must have some preceding history h'_α , where $h_\alpha = h'_\alpha \circ t$ for some action-percept pair t . This definition of EPRU must examine h'_α and h_α independently, even though their only difference is the addition of t .

For this reason, it will make more sense computationally to use a “forward” view of EPRU: we will construct a recursive function on s and h_α which, for each possible $h_\alpha \circ t$:

- calculates the per-round utility gained from t , both for history $h_\alpha \circ t$ and for all histories that can be reached from $h_\alpha \circ t$, and then
- recurses on s and $h_\alpha \circ t$.

For the calculation in the first bullet, we will use the formula $\text{EV}_{exp}(r, v)$, which computes the expected amount of per-round utility we gain (on the current round and on future rounds) by exploiting a value v on round r

$$\text{EV}_{exp}(r, v) = \sum_{j=r}^{\infty} \frac{L(j)v}{j} = v \sum_{j=r}^{\infty} \frac{1}{j} (1-d)^{j-1} \quad (11)$$

Using known properties of infinite series, EV_{exp} can also be expressed as ⁸

$$\begin{aligned} \text{EV}_{exp}(r, v) &= v \left(\sum_{i=1}^{\infty} \frac{1}{i} (1-d)^{i-1} - \sum_{i=1}^{r-1} \frac{1}{i} (1-d)^{i-1} \right) \\ &= v \left(\frac{\ln d}{d-1} - \sum_{i=1}^{r-1} \frac{1}{i} (1-d)^{i-1} \right) \end{aligned} \quad (12)$$

and is therefore computable.

We can now express the expected per-round utility of a strategy s recursively in terms of the average per-round payoff of an agent.

$$\text{EPRU}_{alt}(s, h_\alpha | G, \mathbf{S}) = \sum_{t \in T} P(h_\alpha \circ t | h_\alpha, s(h_\alpha), \mathbf{S}) \cdot (\text{EV}_{exp}(|h_\alpha \circ t|, U(t)) + \text{EPRU}_{alt}(s, h_\alpha \circ t | G, \mathbf{S})) \quad (13)$$

where T is the set of all possible action-percept pairs, and $h_\alpha \circ t$ represents a possible history on the next round. Note that the size of T is finite. A proof that $\text{EPRU}(s | G, \mathbf{S})/d = \text{EPRU}_{alt}(s, \langle \rangle | G, \mathbf{S})$ is included in Appendix A.

⁸The simplification $\sum_{i=1}^{\infty} \frac{1}{i} (1-d)^{i-1} = \frac{\ln d}{d-1}$ is due to [7].

Unfortunately, computing EPRU_{alt} is not possible since it suffers from infinite recursion. To handle this, we introduce a depth-limited computation of EPRU_{alt} , which only computes the portion of the total EPRU contributed by the first k rounds:

$$\text{EPRU}_{\text{alt}}^k(s, h_\alpha | G, \mathbf{S}) = \begin{cases} 0 & \text{If } k = 0 \\ \sum_{t \in T} P(h_\alpha \circ t | h_\alpha, s(h_\alpha), \mathbf{S})(\text{EV}_{\text{exp}}(|h_\alpha|, U(t)) + \text{EPRU}_{\text{alt}}^{k-1}(s, h \circ t | G, \mathbf{S})) & \text{otherwise} \end{cases} \quad (14)$$

We prove in Section 7.3 that if the search depth k is deep enough, $\text{EPRU}_{\text{alt}}^k(s, h_\alpha | G, \mathbf{S})$ will always be within ϵ of $\text{EPRU}_{\text{alt}}(s, h_\alpha | G, \mathbf{S})$.

6.2 EPRU Corresponds to Reproductive Success

This section provides a proof that if a strategy has the highest EPRU for the given environment, it will also have the optimal expected probability of reproducing. This proof applies only to pairwise games, but the same techniques can apply to arbitrary (finite) numbers of strategies.

Assume we have an infinite Cultaptation game instance (G, \mathbf{S}) , as defined in Section 5, made up of agents using strategies s and s' (i.e. $\mathbf{S} = \{s, s'\}$). Recall from Section 5 that the score for strategy s is

$$\text{score}(s) = \lim_{r \rightarrow \infty} \frac{\sum_{i=0}^r p(s, i)}{r}$$

where $p(s, i)$ is the number of agents using strategy s on round i . Our objective for this section will be to show that $\text{EPRU}(s | G, \mathbf{S}) > \text{EPRU}(s' | G, \mathbf{S})$ if and only if $\text{score}(s) > \text{score}(s')$.

We begin by defining a reset event, which will help us illustrate some interesting properties of infinite Cultaptation.

Definition. Let n and n' be the number of agents using s and s' , respectively, on the first round of the game instance, and let $N = n + n'$. A *reset event* occurs when all the agents in the environment die on two consecutive rounds, and on the second round they are replaced (via mutation) by n agents using s and n' agents using s' . The probability of a reset event occurring is $\beta = d^N d^N m^N \binom{n}{N} 0.5^n$. \square

In other words, after a reset event occurs the conditions are identical to those that were present on the first round; the game instance has essentially started over. Note that β is the same on every round, and it is always greater than 0.

Since the game instance continues for an infinite number of rounds, there will be an infinite number of reset events. Thus, if we were to run other game instances with $\mathbf{S} = \{s, s'\}$, both strategies would have the same score each time. Therefore, we also know that we can define each strategy's score as a function of its *expected population* at each round, rather than its population for a single game instance. This gives us

$$\lim_{r \rightarrow \infty} \frac{\sum_{i=0}^r p(s, i)}{r} = \lim_{r \rightarrow \infty} \frac{\sum_{i=0}^r \text{EP}(s, i)}{r} \quad (15)$$

where $\text{EP}(s, r)$ is the expected population of agents using strategy s on round r .

We will also define $\text{EAU}(s, r)$ to be the *expected agent utility* of strategy s on round r ; that is, $\text{EAU}(s, r)$ is the expected PRU of a randomly-chosen agent using strategy s on round r . We can now define $\text{EP}(s, r)$ recursively for each strategy using the mechanics of Cultaptation, as follows. We will let $\text{EP}(s, 0) = n$ and $\text{EP}(s', 0) = n'$. Then, for $r \geq 0$

$$\text{EP}(t, r+1) = \underbrace{(1-d)\text{EP}(t, r)}_{\text{Survived from previous round}} + \underbrace{Nd(1-m)\frac{\text{EP}(t, r)\text{EAU}(t, r)}{\text{TU}(r)}}_{\text{New agents from selection}} + \underbrace{Nd\frac{m}{2}}_{\text{New agents from mutation}}$$

where $t \in \{s, s'\}$ and $TU(r) = EP(s, r)EAU(s, r) + EP(s', r)EAU(s', r)$ is the expected total utility on round r . Recall from Section 5 that N is the total number of agents in the environment, d is the probability of death, and m is the probability of mutation.

We now consider the behavior of $EAU(s, r)$ as r increases.

Lemma 1 *For any strategy s , $\lim_{r \rightarrow \infty} EAU(s, r) = \gamma$ for some finite γ .*

Proof. Let $u(s, r)$ be the expected utility of a single agent using strategy s when r rounds have passed since the first round or the last reset event. For all r , we know that $0 \leq u(s, r) \leq V_{\max}/d$, since agents cannot earn negative utility, and no strategy can do better than exploiting the best possible action for its entire expected lifespan of $1/d$ rounds. We can rewrite $EAU(s, r)$ in terms of $u(s, r)$ as follows.

$$EAU(s, r) = \beta \left(\sum_{i=0}^{r-1} (1-\beta)^i u(s, r) \right) + (1-\beta)^r u(s, r)$$

Taking the limit of this form gives us

$$\lim_{r \rightarrow \infty} EAU(s, r) = \lim_{r \rightarrow \infty} \beta \left(\sum_{i=0}^{r-1} (1-\beta)^i u(s, i) \right) + \lim_{r \rightarrow \infty} (1-\beta)^r u(s, r) = \lim_{r \rightarrow \infty} \beta \left(\sum_{i=0}^{r-1} (1-\beta)^i u(s, i) \right).$$

Since $u(s, i)$ is bounded and $\sum_{i=0}^{r-1} (1-\beta)^i$ is a geometric series, $\lim_{r \rightarrow \infty} \beta \left(\sum_{i=0}^{r-1} (1-\beta)^i u(s, i) \right)$ converges absolutely by the comparison test. Hence, $\lim_{r \rightarrow \infty} EAU(s, r) = \gamma$ for some finite γ . \square

Lemma 2 *For any strategy s_α and set of available strategies \mathbf{S} ,*

$$\lim_{r \rightarrow \infty} EAU(s_\alpha, r) = EPRU(s_\alpha | G, \mathbf{S})$$

Proof. The expected agent utility $EAU(s_\alpha, r)$ is defined as the expected PRU of an agent using strategy s_α on round r . As r approaches infinity, the probability that a randomly-selected agent will be i rounds old approaches $L(i) / \sum_{j=0}^{\infty} L(j) = dL(i)$. The probability of reaching a history h_α is defined in Section 5.5 as $P(h_\alpha | s_\alpha, \mathbf{S})$, and as r increases the set of histories a randomly-selected agent may have approaches H , the set of all histories. Thus,

$$\lim_{r \rightarrow \infty} EAU(s_\alpha, r) = \sum_{h_\alpha \in H} dL(|h_\alpha|) \times P(h_\alpha | s_\alpha, \mathbf{S}) \times PRU(h_\alpha),$$

which is the definition of $EPRU(s_\alpha | G, \mathbf{S})$. \square

$EP(s, r)$ and $EP(s', r)$ are both functions of $EAU(s, r)$ and $EAU(s', r)$, which converge to $EPRU(s | G, \mathbf{S})$ and $EPRU(s' | G, \mathbf{S})$ respectively. Therefore, $EP(s, r)$ and $EP(s', r)$ must also converge as r approaches infinity. We will let $EP(s) = \lim_{r \rightarrow \infty} EP(s, r)$ for $s \in \{s, s'\}$. We can find the value of $EP(s)$ as follows

$$EP(s) = (1-d)EP(s) + Nd(1-m) \frac{EP(s)EPRU(s | G, \mathbf{S})}{EP(s)EPRU(s | G, \mathbf{S}) + EP(s')EPRU(s' | G, \mathbf{S})} + Nd \frac{m}{2}.$$

After substituting $EP(s') = N - EP(s)$ and rearranging terms, we have

$$\begin{aligned} 0 &= (EPRU(s | G, \mathbf{S})EP(s)^2 - EPRU(s' | G, \mathbf{S})) \\ &+ N \left(\left(1 + \frac{m}{2}\right) EPRU(s' | G, \mathbf{S}) - \left(1 - \frac{m}{2}\right) EPRU(s | G, \mathbf{S}) \right) EP(s) - N^2 \frac{m}{2} EPRU(s' | G, \mathbf{S}). \end{aligned}$$

Assume $EPRU(s | G, \mathbf{S}) > 0$ and $EPRU(s' | G, \mathbf{S}) > 0$ and let $x = EPRU(s | G, \mathbf{S}) / EPRU(s' | G, \mathbf{S})$. Then we can rewrite the above as

$$0 = (x - 1)EP(s)^2 + N \left(1 + \frac{m}{2} - x \left(1 - \frac{m}{2} \right) \right) EP(s) - N^2 \frac{m}{2}. \quad (16)$$

This equation, when subject to the constraint $0 \leq EP(s) \leq N$, allows us to express $EP(s)$ as a strictly increasing function of x . It also has the property that when $x = 1$, $EP(s) = EP(s') = N/2$.

Lemma 3 $EP(s) > EP(s')$ if and only if $EPRU(s | G, \mathbf{S}) > EPRU(s' | G, \mathbf{S})$.

Proof. Assume $EPRU(s | G, \mathbf{S}) > EPRU(s' | G, \mathbf{S})$. Then $x > 1$ in Equation 16, and therefore $EP(s) > N/2$, so $EP(s) > EP(s')$. Assume $EP(s) > EP(s')$. Using Equation 16, we know that $x > 1$ and therefore that $EPRU(s | G, \mathbf{S}) > EPRU(s' | G, \mathbf{S})$. Hence, $EP(s) > EP(s')$ if and only if $EPRU(s | G, \mathbf{S}) > EPRU(s' | G, \mathbf{S})$. \square

We can now calculate the ratio between each strategy's score using $EP(s)$ and $EP(s')$.

Lemma 4 For all s and s' ,

$$\lim_{r \rightarrow \infty} \frac{\sum_{i=0}^r EP(s, i)}{r} > \lim_{r \rightarrow \infty} \frac{\sum_{i=0}^r EP(s', i)}{r}$$

if and only if $EP(s) > EP(s')$.

Proof. We know that

$$\frac{\lim_{r \rightarrow \infty} \frac{\sum_{i=0}^r EP(s, i)}{r}}{\lim_{r \rightarrow \infty} \frac{\sum_{i=0}^r EP(s', i)}{r}} = \frac{\lim_{r \rightarrow \infty} \sum_{i=0}^r EP(s, i)}{\lim_{r \rightarrow \infty} \sum_{i=0}^r EP(s', i)}.$$

Since the sequence $b_r = \sum_{i=0}^r EP(s', i)$ is unbounded and strictly increasing, we can use the Stolz-Cesàro theorem to obtain

$$\frac{\lim_{r \rightarrow \infty} \sum_{i=0}^r EP(s, i)}{\lim_{r \rightarrow \infty} \sum_{i=0}^r EP(s', i)} = \frac{\lim_{r \rightarrow \infty} \sum_{i=0}^{r+1} EP(s, i) - \sum_{i=0}^r EP(s, i)}{\lim_{r \rightarrow \infty} \sum_{i=0}^{r+1} EP(s', i) - \sum_{i=0}^r EP(s', i)} = \frac{\lim_{r \rightarrow \infty} EP(s, r)}{\lim_{r \rightarrow \infty} EP(s', r)} = \frac{EP(s)}{EP(s')}.$$

\square

From Equation 15 and Lemmas 3 and 4, we immediately get the following:

Theorem 1 For all s and s' ,

$$\lim_{r \rightarrow \infty} \frac{\sum_{i=0}^r n(s, i)}{r} > \lim_{r \rightarrow \infty} \frac{\sum_{i=0}^r n(s', i)}{r}$$

if and only if $EPRU(s | G, \mathbf{S}) > EPRU(s' | G, \mathbf{S})$.

Therefore, a strategy's expected reproductive success is directly proportional to its EPRU.

6.2.1 Irrelevance of the Initial Strategy Profile

From the fact that EPRU is independent of the initial strategy profile \mathbf{s} , we also get the following corollary which will help us understand some of our experimental results (see Section 9).

Corollary 1 *The initial strategy profile \mathbf{s} of an infinite Cultaptation game instance (defined in Section 5) does not affect the score of any strategy in \mathbf{S} .*

If this seems counterintuitive, consider the following. At the beginning of this section we defined a reset event, in which every agent dies on two consecutive rounds, and all are replaced via mutation so that the population is identical to the initial strategy profile. For each reset event, there will be many similar events in which every agent dies on two consecutive rounds and is replaced via mutation, but in some arrangement different from the initial strategy profile. The probability of this happening is $d^{2N}m^N$, which is greater than 0. In an infinite-length game, such an event will eventually occur with probability 1. After it occurs, the initial strategy profile clearly has no bearing on how the rest of the game plays out, yet there are still an infinite number of rounds left in the game. Since each strategy's score is its average population over the entire game (see Section 5), the impact of the initial strategy profile on each strategy's total score is vanishingly small in an infinite-length game.

6.2.2 Application of EPRU to other Evolutionary Games

Many of the equations used in calculating EPRU involve concepts particular to Cultaptation, such as innovation, observation, and changing action values. However, the general technique we use is to calculate the expected reproductive fitness of an agent on round j , multiply this quantity by the expected proportion of agents that are j rounds old, and sum these quantities to get the expected fitness of an entire population. This should be a useful metric in any evolutionary game in which agents live for more than one generation and reproduce according to the replicator equation, even if the game uses some measure other than per-round utility to determine reproductive fitness. The proofs in this section rely primarily on the symmetry between 1) the probability that an agent will be alive after k rounds and 2) the expected proportion of a population of agents that are k rounds old on any given round. Thus, any evolutionary game that allows agents to live more than one generation and in which agents die with the same probability on every round should be able to use a metric very similar to EPRU to compare strategies.

7 Finding an ϵ -Best Response Strategy

In this section we explain what it means for a strategy to be a best response or near-best response in infinite Cultaptation, and we provide an algorithm for calculating a near-best response to $\mathbf{S}_{-\alpha}$, the available strategies other than our own.

7.1 Problem Specification

Now that we have derived EPRU and proved that a strategy's EPRU is directly proportional to its score in an infinite Cultaptation game, we can determine how each strategy in a given set of available strategies \mathbf{S} will perform by evaluating the EPRU of each strategy. Therefore, we can define a best-response strategy in terms of EPRU, as follows.

Recall that in an infinite Cultaptation game (as defined in Section 5) there are ℓ players, each of whom selects a strategy to put into the set of available strategies \mathbf{S} . Let $\mathbf{S}_{-\alpha}$ be the set of available strategies other

than our own, i.e. $\mathbf{S}_{-\alpha} = \{s_1, \dots, s_{\alpha-1}, s_{\alpha+1}, \dots, s_\ell\}$. We will say that strategy s_{opt} is a *best response* to $\mathbf{S}_{-\alpha}$ if and only if for any other strategy s' ,

$$\text{EPRU}(s_{\text{opt}} | G, \mathbf{S}_{-\alpha} \cup s_{\text{opt}}) \geq \text{EPRU}(s' | G, \mathbf{S}_{-\alpha} \cup s').$$

Computing s_{opt} is not possible due to its prohibitively large size. However, we can compute an ϵ -*best-response strategy*, i.e., a strategy s such that $\text{EPRU}(s | G, \mathbf{S}_{-\alpha} \cup s)$ is arbitrarily close to $\text{EPRU}(s_{\text{opt}} | G, \mathbf{S}_{-\alpha} \cup s_{\text{opt}})$. This problem can be stated formally as follows: Given game parameters G , error bound $\epsilon > 0$, and the set \mathbf{S} of available strategies other than our own, find a strategy s_α such that $\text{EPRU}(s_\alpha | G, \mathbf{S}_{-\alpha} \cup s_\alpha)$ is within ϵ of $\text{EPRU}(s_{\text{opt}} | G, \mathbf{S}_{-\alpha} \cup s_{\text{opt}})$.

7.2 Bounding EPRU

In games where $0 < d < 1$, an agent could potentially live for any finite number of rounds. However, since the agent's probability of being alive on round r decreases exponentially with r , the expected utility contributed by an agent's actions in later rounds is exponentially lower than the expected utility contributed by earlier rounds. We will use this fact in deriving a bound on $\text{EPRU}_{\text{alt}}(s, h_\alpha | G, \mathbf{S})$ for a given strategy and a history h_α of length l .

Recall from Equations 11 and 12 that:

$$\text{EV}_{\text{exp}}(r, v) = v \sum_{i=r}^{\infty} \frac{1}{i} (1-d)^{i-1} = v \left(\frac{\ln d}{d-1} - \sum_{i=1}^{r-1} \frac{1}{i} (1-d)^{i-1} \right) \quad (17)$$

where $\text{EV}_{\text{exp}}(r, v)$ is the expected contribution to EPRU made by exploiting an action with value v on round r .

Since we know how much any given exploit contributes to the expected $\text{EPRU}(s_\alpha | G, \mathbf{S})$ for a given strategy s_α , we can calculate $G(l, v)$, the amount that exploiting the same action on *all* rounds after l contributes to $\text{EPRU}(s_\alpha | G, \mathbf{S})$, as follows:

$$G(l, v) = \sum_{j=l+1}^{\infty} v \sum_{n=j}^{\infty} \frac{1}{n} (1-d)^{n-1} = v \sum_{j=l+1}^{\infty} \sum_{n=j}^{\infty} \frac{1}{n} (1-d)^{n-1}$$

Expanding the summations yields:

$$\begin{aligned} G(l, v) &= v \left(\frac{1}{l+1} (1-d)^l + \frac{2}{l+2} (1-d)^{l+1} + \dots \right) \\ &= v \sum_{n=l+1}^{\infty} \frac{n-l}{n} (1-d)^{n-1} \\ &= v \left(\sum_{n=l+1}^{\infty} (1-d)^{n-1} - \sum_{n=l+1}^{\infty} \frac{l}{n} (1-d)^{n-1} \right) \\ &= v \left(\frac{(1-d)^l}{d} - \sum_{n=l+1}^{\infty} \frac{l}{n} (1-d)^{n-1} \right) \end{aligned}$$

Next, we pull l out of the summation and use (17) to obtain:

$$G(l, v) = v \left(\frac{(1-d)^l}{d} - l \underbrace{\left(\frac{\ln d}{d-1} - \sum_{n=1}^l \frac{1}{n} (1-d)^{n-1} \right)}_a \right) \quad (18)$$

Note that for $0 < d < 1$, $G(l, v)$ is finite. $G(l, v)$ provides a closed form formula for the eventual contribution of exploiting an action with value v at every round after the l^{th} round. Since the set V of possible action values is finite (see Section 5), let $v_{max} = \max(V)$ be the largest of these values. Then $G(l, v_{max})$ is an upper bound on the expected per-round utility achieved after round l (clearly no strategy can do better than making an action with maximal value every action after action l). We use this fact to bound the depth limited expected per-round utility computation.

Theorem 2 *Let v_{max} be the highest possible action utility for game parameters G , and let $\mathbf{S}_{-\alpha}$ be the set of available strategies other than our own. Then for all l and all strategies s_α ,*

$$\text{EPRU}_{\text{alt}}(s_\alpha, \langle \rangle \mid G, \mathbf{S}_{-\alpha} \cup s_\alpha) - \text{EPRU}_{\text{alt}}^l(s_\alpha, \langle \rangle \mid G, \mathbf{S}_{-\alpha} \cup s_\alpha) \leq G(l, v_{max}).$$

Proof. Since it is not possible for any strategy to gain more utility than v_{max} on any round, this follows from the discussion above. \square

Theorem 2 states that $G(l, v_{max})$ is the highest possible contribution to $\text{EPRU}_{\text{alt}}(s_\alpha, \langle \rangle \mid G, \mathbf{S})$ made by any strategy s_α after round l . Thus, if we are given an $\epsilon > 0$ and we can find a value of k such that $G(k, v_{max}) \geq \epsilon$, then we know that no strategy can earn more than ϵ expected utility after round k . The next section will show how to find such a k .

7.3 Determining How Far to Search

In this section we show how to find a search depth k such that, for any given $\epsilon > 0$, no strategy can earn more than ϵ utility after round k . We first note a bound on $G(l, v)$:

Lemma 5 $G(l, v) \leq v(1 - d)^l/d$.

Proof. The lemma follows from noting that part (a) of Equation 18 is greater than or equal to zero, since $\frac{\ln d}{d-1} = \sum_{n=1}^{\infty} \frac{1}{n}(1-d)^{n-1}$ and $l < \infty$. Thus $G(l, v) = v\left(\frac{(1-d)^l}{d} - w\right) \leq v\frac{(1-d)^l}{d}$, since w is always non-negative. \square

Now if we can find a k such that

$$\epsilon = v_{max}(1 - d)^k/d,$$

then we can be certain that $\epsilon \geq G(k, v)$. Solving for k in the above equation yields

$$k = \log_{(1-d)}\left(\frac{d\epsilon}{v_{max}}\right), \tag{19}$$

which has a solution for $0 < d < 1$ and $v_{max} > 0$, both of which will always be true in Cultaptation. This gives us the following theorem.

Theorem 3 *Given $\epsilon > 0$, set of available strategies $\mathbf{S}_{-\alpha}$ other than our own, and game parameters G with maximal utility v_{max} , let $k = \log_{(1-d)}\left(\frac{d\epsilon}{v_{max}}\right)$. If s_α has the maximal value of $\text{EPRU}_{\text{alt}}^k(s_\alpha, \emptyset \mid G, \mathbf{S}_{-\alpha} \cup s_\alpha)$, then s_α is an ϵ -best response to $\mathbf{S}_{-\alpha}$.*

Proof. Let s_{opt} be the strategy with the maximal value of $\text{EPRU}(s_{\text{opt}} \mid G, \mathbf{S}_{-\alpha} \cup s_{\text{opt}})$. By Theorem 2, we know that s_{opt} cannot earn more than ϵ expected utility on rounds after k . Since s_α earns the maximum EPRU possible in the first k rounds, it follows that $|\text{EPRU}(s_{\text{opt}} \mid G, \mathbf{S}_{-\alpha} \cup s_{\text{opt}}) - \text{EPRU}(s_\alpha \mid G, \mathbf{S}_{-\alpha} \cup s_\alpha)| \leq \epsilon$. Therefore, s_α is an ϵ -best response. \square

Algorithm 1 Produce strategy s that maximizes $\text{EPRU}_{\text{alt}}^k(s, h_\alpha \mid G, \mathbf{S}_{-\alpha} \cup s)$, given initial history h_α , set of possible utility values V , and $\mathbf{S}_{-\alpha}$, the set of available strategies other than our own.

$\text{Strat}(h_\alpha, k, V, \mathbf{S}_{-\alpha})$

```

1: if  $k = 0$  then
2:   return 0
3: end if
4: Let  $U_{\max} = 0$ 
5: Let  $s_{\max} = \text{null}$ 
6: for each action  $a \in \{\text{Inv}, \text{Obs}, X_1, \dots, X_\mu\}$  do
7:   Let  $U_{\text{temp}} = 0$ 
8:   Let  $s_{\text{temp}} = \langle h_\alpha, a \rangle$ 
9:   for each action  $m \in \{1, \dots, \mu\}$  do
10:    for each value  $v \in V$  do
11:      Let  $t = (a, (m, v))$ 
12:      Let  $p = P(h_\alpha \circ t \mid h_\alpha, a, \mathbf{S}_{-\alpha})$ 
13:      if  $p > 0$  then
14:        Let  $\{S', U'\} = \text{Strat}(h_\alpha \circ t, k - 1, V, \mathbf{S}_{-\alpha})$ 
15:         $s_{\text{temp}} = s_{\text{temp}} \cup S'$ 
16:         $U_{\text{temp}} = U_{\text{temp}} + p(\text{EV}_{\text{exp}}(|h_\alpha \circ t|, U(t)) + U')$ 
17:      end if
18:    end for
19:  end for
20:  if  $U_{\text{temp}} > U_{\max}$  then
21:     $U_{\max} = U_{\text{temp}}$ 
22:     $s_{\max} = s_{\text{temp}}$ 
23:  end if
24: end for
25: return  $\{s_{\max}, U_{\max}\}$ 

```

7.4 Algorithm

We will now present our algorithm for computing the strategy s with the maximal value of $\text{EPRU}_{\text{alt}}^k(s, \emptyset \mid G, \mathbf{S}_{-\alpha} \cup s)$, and show how it can be used to compute an ϵ -best response.

Algorithm 1 returns a 2-tuple with a partially specified strategy s and a scalar U . Strategy s maximizes $\text{EPRU}_{\text{alt}}^k(s, h_\alpha \mid G, \mathbf{S}_{-\alpha} \cup s)$, and U is the value of this expression.

The algorithm performs a depth-first search through the space of strategies that start from the input history h_α , stopping once it reaches a specified depth k . For each possible action $a \in \{\text{Inv}, \text{Obs}, X_1, \dots, X_\mu\}$ at h_α , it computes the expected per-round utility gained from performing a , and the utility of the best strategy for each possible history h'_α that could result from choosing a . It combines these quantities to get the total expected utility for a , and selects the action with the best total expected utility, a_{\max} . It returns the strategy created by combining the policy $\langle h_\alpha, a_{\max} \rangle$ with the strategies for each possible h'_α , and the utility for this strategy.

Seen another way, $\text{Strat}(h_\alpha, k, V, \mathbf{S}_{-\alpha})$ computes $\text{EPRU}_{\text{alt}}^k(s, h_\alpha \mid G, \mathbf{S}_{-\alpha} \cup s)$ for all possible strategies s , returning the strategy maximizing $\text{EPRU}_{\text{alt}}^k$ as well as the maximal value of $\text{EPRU}_{\text{alt}}^k$.

Proposition 1 $\text{Strat}(h_\alpha, k, V, \mathbf{S}_{-\alpha})$ returns (s, U) such that

$$\text{EPRU}_{\text{alt}}^k(s, h_\alpha \mid G, \mathbf{S}_{-\alpha} \cup s) = U = \text{argmax}_{s'}(\text{EPRU}_{\text{alt}}^k(s', h_\alpha \mid G, \mathbf{S}_{-\alpha} \cup s')).$$

A proof of this proposition is presented in Appendix A.

We now have an algorithm capable of computing the strategy with maximal expected utility over the first k rounds. Hence, in order to find an ϵ -best response strategy we need only find the search depth k such that no strategy can earn more than ϵ expected utility after round k , and then call the algorithm with that value of k .

Theorem 4 *Given $\epsilon > 0$, available strategies other than our own $\mathbf{S}_{-\alpha}$, and a set of values V with maximum value v_{max} , let $k = \log_{(1-d)}\left(\frac{d\epsilon}{v_{max}}\right)$. Then $\text{Strat}(\emptyset, k, V, \mathbf{S}_{-\alpha})$ returns (s, U) such that s is an ϵ -best response to $\mathbf{S}_{-\alpha}$.*

Proof. This follows from Theorem 3 and Proposition 1. □

We also have the following.

Corollary 2 *Given available strategies other than our own $\mathbf{S}_{-\alpha}$ and a set of values V , let s_k be the strategy returned by $\text{Strat}(\emptyset, k, V, \mathbf{S}_{-\alpha})$. Then $\lim_{k \rightarrow \infty} s_k$ is a best response to $\mathbf{S}_{-\alpha}$.*

Proof. Let s_{opt} be a best response to $\mathbf{S}_{-\alpha}$. By Lemma 5 and Theorem 3,

$$\text{EPRU}(s_{\text{opt}} | G, \mathbf{S}_{-\alpha} \cup s_{\text{opt}}) - \text{EPRU}(s_k | G, \mathbf{S}_{-\alpha} \cup s) \leq v_{max}(1-d)^k/d.$$

Since $\lim_{k \rightarrow \infty} v_{max}(1-d)^k/d = 0$, it follows that $\lim_{k \rightarrow \infty} (\text{EPRU}(s_{\text{opt}} | G, \mathbf{S}_{-\alpha} \cup s_{\text{opt}}) - \text{EPRU}(s_k | G, \mathbf{S}_{-\alpha} \cup s_k)) = 0$. Therefore, $\lim_{k \rightarrow \infty} s_k$ is a best response to $\mathbf{S}_{-\alpha}$. □

7.5 Implementation

In this section we discuss modifications that improve the running time of Algorithm 1 without any loss in accuracy. Section 7.5.1 discusses techniques for state aggregation, which allow us to cut the branching factor of the algorithm in half. Section 7.5.2 discusses the representation of π_{Obs} , and Section 7.5.3 discusses caching and pruning.

7.5.1 State Aggregation

If the pseudocode for Algorithm 1 were implemented verbatim, it would search through each history that can be reached from the starting state. However, there is a significant amount of extraneous information in each history that is not needed for any of the algorithm's calculations. For example, the histories $h_\alpha = \langle\langle \text{Inv}, (1, 10) \rangle\rangle$ and $h'_\alpha = \langle\langle \text{Inv}, (2, 10) \rangle\rangle$ both describe a situation where α innovates once and obtains an action with value 10. The only difference between these histories is the identifier assigned to the action, which does not impact any of the calculations—yet the pseudocode must still search through each of these histories separately. We can eliminate this redundancy by using *repertoires*, rather than histories, as the states for the algorithm to search through. A repertoire is a record of what the agent knows about each of the actions it has learned, rather than a record of everything that has happened to it.

Making this simple change allows Algorithm 1 to calculate the value of an observation action by combining information it learns when exploring innovate and exploit actions, rather than recursing again. This cuts the branching factor of our search in half. The analysis and details involved in this change, as well as the proof that the version of the algorithm using repertoires returns the same result as the previous version, are included in Appendix B.

Running time analysis. When Algorithm 1 considers a history h_a , it makes one recursive call for each possible action-percept pair $(a, (m, v))$ that can be executed at h_a . There are 2μ such pairs for each history; if our agent knows how to exploit j actions, then it can innovate any of the $\mu - j$ actions it does not know, and it can observe any of the μ actions. Each of these actions can also have any of v values. Hence, the number of recursive calls made by the algorithm each action is at most $2\mu v$. Since the algorithm recurses to depth k , the running time for Algorithm 1 is $O((2\mu v)^k)$. With the state aggregation technique described above, we do not need to perform additional recursions for observation actions. Hence, the number of recursive calls made each action is at most μv , and the total running time is $O((\mu v)^k)$, which improves upon the original running time by a factor of 2^k .

7.5.2 Representing π_{Obs}

For our formal proofs, we treated π_{Obs} as a black box that, when given our agent’s history and round number, could tell us the exact probabilities of observing each action on the current round. However, since there are an exponential number of possible histories, storing π_{Obs} in this form would require an exponential amount of space, which would severely limit the size of games for which we could compute strategies. We would also need to run a prohibitively large number of simulations in Algorithm 2 (introduced in Section 8) to get enough samples to generate a new π_{Obs} of this type.

Therefore, as an approximation, our implementation assumes that π_{Obs} has a similar structure to π_{Inv} , and remains constant throughout the agent’s lifetime. That is, the π_{Obs} used in the experiments returns the probability of an action valued v being observed. While this leads to some loss in accuracy, it is very easy to store and compute. Further, we will see in our experimental results (particularly those dealing with iterative computation in Section 9.2) that this form of π_{Obs} is still able to produce good strategies.

7.5.3 Caching and Pruning

Since our implementation uses repertoires rather than histories to represent the agent’s set of known actions, and since it is possible for two histories to produce the same repertoire, the algorithm will sometimes encounter repertoires that it has already evaluated. So that the algorithm will not have to waste time evaluating them again, the implementation includes a cache which stores the EPRU of every repertoire it has evaluated. When the algorithm encounters a repertoire whose expected utility is needed, the implementation first checks the cache to see if the EPRU of the repertoire has been previously computed, and uses the computed value if it exists. Caching is widely used in tree-search procedures, and is analogous to the transposition tables in chess-playing algorithms [31].

We also use another well-known method for avoiding unnecessary evaluation of states, namely branch-and-bound pruning [19, 17]. Before one computes the expected per-round utility of a given action, one checks to see if an upper bound on the EPRU of that action would be sufficient to make the given action’s utility higher than the best previously computed action. In many situations, the maximal utility that can be achieved for a given action will in fact be less than the utility we know we can achieve via some other action, and therefore we can skip the evaluation of that action (i.e. we can “prune” it from the search tree).

We have no theoretical guarantees on runtime reduction using these techniques, but we will see in Section 9.1.1 that the combination of pruning and caching allows us to avoid evaluating significant portions of the state space in the environments we tested.

8 Cultaptation Strategy Learning Algorithm

Until now we have assumed that Algorithm 1 has access to π_{Obs} , the distribution of observable actions, when it performs its calculations. While the algorithm finds the near-best-response strategy given a particular π_{Obs} ,

Algorithm 2 Produce an approximation of a strategy that is an ϵ -best response to itself.

CSLA($\pi_{\text{Inv}}, \tau, k$)

- 1: Let $\pi_{\text{Obs}} = \pi_{\text{Inv}}$.
 - 2: $s = \emptyset$.
 - 3: **repeat**
 - 4: Let $s_{\text{old}} = s$.
 - 5: Let $V = \{\pi_{\text{Inv}}, \pi_{\text{Obs}}\}$.
 - 6: $s = \text{Strat}(\emptyset, k, V, \mathbf{S})$
 - 7: Simulate a series of Cultaptation games in which s plays itself, and action utilities are initially drawn from π_{Inv} , recording all actions exploited in the last quarter of this game.
 - 8: Use records of exploited actions to generate a new distribution π_{Obs} (i.e. $\pi_{\text{Obs}}(v) =$ fraction of the time v was exploited in the records).
 - 9: **until** $\text{stratDiff}(s, s_{\text{old}}) < \tau$
 - 10: **return** s .
-

agents playing the real Cultaptation game are not given access to π_{Obs} beforehand, and even estimating what π_{Obs} looks like can be very difficult while playing the game due to the limited amount of information each agent receives in its lifetime. It is also unclear how exactly an agent’s own actions will affect π_{Obs} : by exploiting a particular action, the agent is making that action observable to others who might then exploit it in greater proportion than in the π_{Obs} used to compute the agent’s strategy.

To address these difficulties, we developed the Cultaptation Strategy Learning Algorithm (CSLA), which uses a method for creating a strategy and a distribution π_{Obs} *simultaneously* so that (i) π_{Obs} is the distribution created when all agents in a Cultaptation game play the computed strategy and (ii) the computed strategy is a near-best response for π_{Obs} (and other parameters).

This algorithm copes with the lack of information about π_{Obs} , and generates an approximation of a strategy that is a best response to itself. At a high level, the algorithm can be thought of as generating a series of strategies, each an ϵ -best response to the one before it, and stopping when two successive strategies are extremely similar. A more detailed description of this process follows.

The algorithm begins by assuming $\pi_{\text{Obs}} = \pi_{\text{Inv}}$. The algorithm then proceeds iteratively; at each iteration it generates s , the ϵ -best response strategy to the current π_{Obs} , then simulates a series of Cultaptation games in which s plays itself, and extracts a new π_{Obs} from the actions exploited in these games.

At the end of each iteration, the algorithm compares s to s_{old} , the strategy produced by the previous iteration, using the stratDiff function. $\text{stratDiff}(s, s_{\text{old}})$ computes the probability that an agent using s would perform at least one different action before dying than the same agent using s_{old} . For instance, $\text{stratDiff}(s, s_{\text{old}}) = 1.0$ means that the two strategies will always perform at least one different action (i.e. the actions they choose on the first round are different), while $\text{stratDiff}(s, s_{\text{old}}) = 0.0$ means that s is identical to s_{old} .

When $\text{stratDiff}(s, s_{\text{old}})$ is found to be below some threshold τ , CSLA terminates and returns s , the strategy computed by the last iteration. The formal algorithm is presented as Algorithm 2.

Properties of the strategy. In our experimental studies (see Section 9), the strategies produced by CSLA in any given game were all virtually identical, even when a random distribution (rather than π_{Inv}) was used to initialize π_{Obs} . This strongly suggests (though it does not prove) that the strategy profile consisting of copies of s_{self} is a symmetric near-Nash equilibrium.

Furthermore, there is reason to believe that s is evolutionarily stable. Consider an environment in which all agents use the strategy s , and suppose a small number (say, one or two) other strategies are introduced as invaders. Because s was an near-best response to the environment that existed before the opponent’s

agents are introduced, and because the introduction of one or two invaders will change this environment only slightly, agents using s will still be using a strategy that is close to the best response for the current environment, and they will also have some payoff they have accumulated on previous rounds when their strategy was still a near-best response. Thus, the invaders should have a difficult time establishing a foothold in the population, hence should die out with high probability. This suggests (but does not prove) that s is evolutionarily stable.⁹

8.1 Implementation Details

We have created a Java implementation of CSLA. Here we briefly discuss two issues we dealt with during implementation.

8.1.1 Representing π_{Obs}

Our implementation of CSLA uses the same representation of π_{Obs} as our implementation of Algorithm 1 does. In other words, it assumes π_{Obs} has the same form as π_{Inv} , and remains constant throughout the game. Ideally we would be able to condition π_{Obs} on the agent’s history, but in practice this would require too much space (since there are an exponential number of possible histories), and we would need to run too many simulations in step 7 to get an accurate distribution for each history.

8.1.2 Training

In the Machine Learning literature, the process of improving an agent’s performance on a given task is often referred to as “training.” In Algorithm 2, strategy s is trained by playing against itself in a series of simulated games in step 7. However, in our implementation of CSLA we have left the agents involved in the games in step 7 as a parameter to the algorithm. This means that CSLA can also produce a strategy that is trained by playing in an environment consisting of itself and one or more given strategies. The intuition behind this approach is that a strategy trained by playing against itself and strategy s' may perform better when playing against s' than a strategy trained against itself alone. We test this hypothesis experimentally, in Section 9.2.

9 Experimental Results

In this section we present our experimental results.

Section 9.1.1 examines the performance of our implementation of the ϵ -best response algorithm. We find that our optimizations allow the algorithm to find strategies within 1% of the best response 1,000 times faster than the unoptimized algorithm. Section 9.1.2 examines the strategies found by the ϵ -best response algorithm when presented with different environments and strategy profiles, and the results give us an idea of what kinds of circumstances are necessary for the near-best-response strategy to prefer innovation over observation.

Section 9.2 presents a series of experiments comparing two strategies generated with our Cultaptation Strategy Learning Algorithm to a known good strategy used in the international Cultaptation tournament. We find that the strategies generated with CSLA are able to beat the known good strategy, even when the environment is different than the one CSLA used to learn the strategies (Sections 9.2.2 and 9.2.3). Finally, we perform an in-depth qualitative analysis of all three strategies and highlight the differences in behavior that give our learned strategies an advantage (Section 9.2.4).

⁹Among other things, a formal proof would require a way to calculate the payoffs for s and any invading strategy. Accomplishing this is likely to be complicated, but we hope to do it in our future research.

Table 4: Expected per-round utility of the ϵ -best response strategy computed by Algorithm 4, for eight different values of ϵ in various environments.

	$\epsilon = 0.4$	$\epsilon = 0.8$	$\epsilon = 1.2$	$\epsilon = 1.6$	$\epsilon = 2.0$	$\epsilon = 2.4$	$\epsilon = 2.8$	$\epsilon = 3.2$
EPRU in uniform1	40.1	40.0	39.6	39.6	39.6	38.9	38.9	38.9
EPRU in uniform10	39.3	39.1	38.8	38.8	38.8	38.1	38.1	38.1
EPRU in uniform20	38.7	38.5	38.2	38.2	38.2	37.7	37.7	37.7
EPRU in uniform30	38.7	38.5	38.2	38.2	38.2	37.7	37.7	37.7
EPRU in uniform40	38.7	38.5	38.2	38.2	38.2	37.7	37.7	37.7

9.1 Experiments with ϵ -Best Response Algorithm

In this section we present the experiments involving our implementation of Algorithm 4, which generates ϵ -best-response strategies for a given set of game parameters and strategy profile.

9.1.1 Implementation Performance

Our first set of experiments was designed to study the accuracy and running time of our implementation, and the effectiveness of the methods we have developed to improve its performance. We first examined the effect of ϵ on running time and on the expected per-round utility computed by Algorithm 4. We ran the experiments in several different environments: first we examined the `uniform1` environment. In this environment, π_{Inv} is a uniform distribution over the values in $\{33.33, 66.67, 100, 133.33, 166.67\}$, and \mathbf{S} contains the innovate-once (I1) strategy from Example 1, so π_{Inv} is identical to π_{Obs} . The probability of change in `uniform1` is 1%, and the probability of death is 40%.

We also introduced several variations on the `uniform1` environment to study the effect of different probabilities of change. They are `uniform10`, `uniform20`, `uniform30`, and `uniform40`, which have the respective probabilities of change of 10%, 20%, 30%, and 40%.

In Table 4, we see the EPRU computed for various values of epsilon in these environments. As a point of reference, strategy I1 can be analytically shown to achieve an EPRU of about 38.56. We can see that an upper bound on achievable EPRU in the `uniform1` environment is 40.5, since the EPRU of an ϵ -best response to \mathbf{S} is 40.1 when ϵ is 0.4. Also, we note that the algorithm finds lower EPRUs as the probability of change increases. This is as expected: in a rapidly changing environment, one cannot expect an agent to do as well as in a static environment where good actions remain good and bad actions remain bad. The ϵ -best-response strategies computed generally innovate as the first action, then exploit that value if it is not the lowest value available (in this case 33.33). Otherwise, the strategies tend to innovate again in an attempt to find an action with a value bigger than 33.33. This is how they manage to achieve a higher EPRU than the innovate-once strategy.

As part of the experiment in the `uniform1` environment, we kept track of the number of nodes searched by four variations of the algorithm. In the first variation, we ran $\text{Strat}(R, r, k, V, \mathbf{S})$ (Algorithm 4) without optimizations. We also examined the algorithm’s performance with the pruning and caching optimizations described in Section 7.5.3.

In Figure 2 we see that employing both caching and pruning allows us to compute strategies within 1% of the best response about 1,000 times faster. We note that the search times required for 80,000-node search is around 15 seconds on a 3.4GHz Xeon processor.

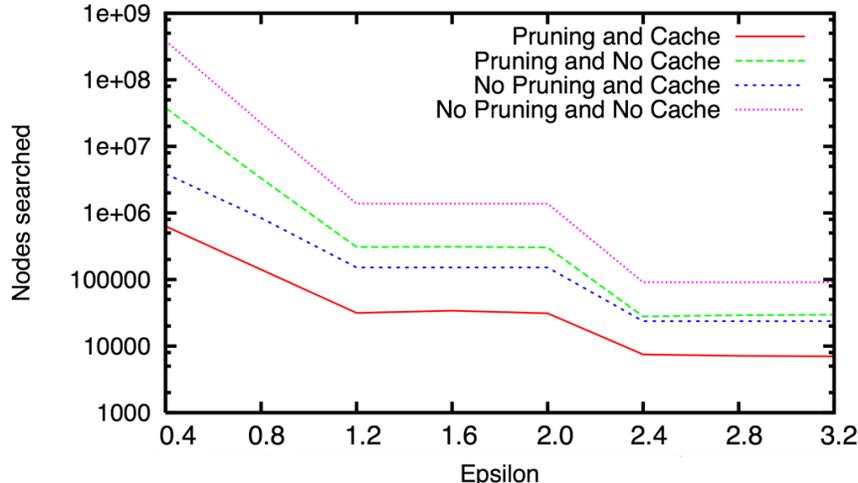


Figure 2: Number of nodes searched in the uniform1 environment, with different combinations of caching and pruning.

9.1.2 Effects of Varying π_{Inv} and π_{Obs} on the ϵ -Best-Response

The objective of this experiment was to study how near-best-response strategies (as computed by Algorithm 4) change as we vary the mean and standard deviation of π_{Inv} and π_{Obs} (which we will call μ_{Inv} , σ_{Inv} , μ_{Obs} and σ_{Obs} , respectively). If we assume that the other agents in the game are rational and not trying to deceive us by intentionally exploiting low-utility actions, then one should expect that $\mu_{\text{Obs}} \geq \mu_{\text{Inv}}$. It may seem natural, then, to conclude that an agent should choose to observe rather than innovate whenever possible, since the average action returned by observing will have higher utility than one returned by innovating. However, previous work has suggested that the standard deviation of these distributions may also play a role in determining which is better [32]. Also, as discussed in Section 3.1, it is possible to imagine pathological scenarios where a population that relies too heavily on observation can become stuck exploiting a low-value action. We designed this experiment to test the hypothesis that, even if we let $\mu_{\text{Obs}} > \mu_{\text{Inv}}$, we can still vary the standard deviations of these distributions such that the ϵ -best-response strategy computed by $\text{EPRU}_{\text{alt}}^k$ will choose to innovate rather than observe. Our methods and results are presented below.

We used the repertoire-based algorithm $\text{Strat}(R, r, k, V)$ (Algorithm 4) to compute ϵ -best-response strategies for Cultaptation games with several different parameter settings, then analyzed the strategies to determine how often they would observe or innovate. In this experiment, the agents died with 40% probability on each round ($d = 0.4$) and there were 5 potential exploitation actions. These games are smaller than the Cultaptation game used in the tournament, to ensure they can be solved in a reasonable amount of time. In our games, we used distributions π_{Obs} and π_{Inv} with means of 110 and 100 respectively.

Table 5 shows the results for four combinations of parameter settings: $\sigma_{\text{Inv}} \in \{10, 300\}$ and $\sigma_{\text{Obs}} \in \{10, 300\}$. When $\sigma_{\text{Inv}} = 10$, the near-best-response strategy will observe almost exclusively (innovating only in rare cases where observation returns several low-quality moves in a row). However, in the environment where $\sigma_{\text{Inv}} = 300$ the near-best-response strategy includes significantly more Innovates; when $\sigma_{\text{Obs}} = 10$ it will innovate 99.5% of the time, and even when $\sigma_{\text{Obs}} = 300$ it still innovates 21.5% of the time.

This experiment lets us conclude that the means of π_{Inv} and π_{Obs} are *not sufficient to determine if innovation or exploitation is better*. In particular, if the standard deviation of innovated values is high, then innovation becomes more valuable because multiple innovations tend to result in a higher valued action than multiple observations.

An interesting strategy emerges when π_{Inv} and π_{Obs} both have high standard deviations. Even though

Table 5: The portion of innovation actions (calculated as $n_{\text{Inv}}/(n_{\text{Inv}} + n_{\text{Obs}})$) in the ϵ -best-response strategy when the standard deviations of π_{Inv} and π_{Obs} are as specified. In all cases, $\mu_{\text{Inv}} = 100$ and $\mu_{\text{Obs}} = 110$.

	$\sigma_{\text{Obs}} = 10$	$\sigma_{\text{Obs}} = 300$
$\sigma_{\text{Inv}} = 10$	2.80×10^{-9}	3.40×10^{-10}
$\sigma_{\text{Inv}} = 300$	0.995	0.215

the mean value of innovated actions is lower than the mean value of observed actions, the ϵ -best-response strategy in these cases innovates initially, then, if the value innovated is high, exploits that value. If the innovated value is not high, an observation action is performed to ensure the agent has a reasonably-valued action available to exploit until it dies.

9.2 Experiments with the Cultaptation Strategy Learning Algorithm

The objective of our second experiment was to examine the performance of strategies produced by the Cultaptation Strategy Learning Algorithm (Algorithm 2 in Section 8), and the importance of the environment (see Section 8.1.2) used to train these strategies. Specifically, we were interested in—

- examining whether the strategies produced with CSLA were capable of beating a strategy that is known to do well;
- examining whether strategies produced by CSLA were able to perform well in environments different from those they were trained in;
- comparing how well a strategy that is trained only against itself (i.e. all agents in the simulated game in Step 6 of the CSLA algorithm use strategy s) can do at repelling an invader, versus how well a strategy trained against the invader (i.e. the invading strategy is included in the population of agents at Step 6) can do at repelling the invader.

For the previous experiments, we assumed we had an oracle for π_{Obs} . For the rest of this section we will be running experimental simulations, so our oracle will observe what the agents do in the simulations and construct π_{Obs} from this, as described in Section 8.

For the known good strategy we used an algorithm called EVChooser, which performs a few innovation and observation actions early in the game and uses the results of these actions (along with a discount factor) to estimate the expected value of innovating, observing, and exploiting, making the action with the highest expected value. It placed 15th out of over 100 entries in the Cultaptation tournament [5]. We chose EVChooser because (1) it has been shown to be a competitive strategy, (2) since we had written it, its source code was readily available to us (unlike the other successful strategies from the Cultaptation tournament), and (3) it could be tuned to perform well in the Cultaptation environments we used (which, in order to accommodate CSLA’s exponential running time, were much smaller than those used in the international Cultaptation tournament).

For games as small as the ones in our experiments, we believe EVChooser is representative of most of the high-performing strategies from the tournament. Nearly all of the strategies described in the tournament report [5] spend some time trying to figure out what the innovate and observe distributions look like, and afterwards use some heuristic for choosing whether to innovate, observe, or exploit their best known action on any given round. This heuristic often involves some time of expected-value computation; for instance, the winning strategy *discountmachine* used a discount factor to compare the utility gained by exploiting the current best-known action to the utility of possibly learning a better action and exploiting it on all future

rounds, which is exactly what EVChooser does.¹⁰ Unlike our CSLA algorithm, none of the strategies in the tournament conducted lookahead search.

For this experiment, we used an environment where π_{Inv} was a uniform distribution over the actions {20, 40, 80, 160}, probability of change was 1%, and probability of death was 25%. Due to the exponential running time of our strategy generating algorithm, this is the largest environment (i.e. smallest probability of death, highest number of actions and action values) for which we could compute full strategies.

9.2.1 Convergence and Consistency of CSLA

We developed a Java implementation of Algorithm 2 that allows us to specify the type of game to be used for the simulation in Step 7. We then created two strategies, s_{self} and s_{EVC} . The training process for both strategies began with s_0 , the best-response to a random π_{Obs} distribution, and continued by constructing a strategy s_{i+1} as a best-response to the π_{Obs} generated by simulating games involving s_i . When training s_{self} the simulated games consisted solely of agents using s_i , but while training s_{EVC} they consisted of a population of agents using s_i being invaded by EVChooser. In both cases, 100 games were simulated at each step of the iteration, to limit the amount of noise in the π_{Obs} that was extracted from the simulations.

While we have no theoretical guarantees that the strategies produced by Algorithm 2 will converge, the algorithm’s similarity to policy iteration [18] led us to suspect that they would converge. Also, since CSLA is greedy, i.e., it selects the best response strategy at each step of the iteration, we were interested in seeing whether the strategy it found represented a local maximum or a global one.

We designed a simple experiment to see how these issues would play out when generating s_{self} and s_{EVC} : we modified the program to use a randomly-generated distribution for the initial value of π_{Obs} , rather than always initially setting $\pi_{\text{Obs}} = \pi_{\text{Inv}}$ as is done in Algorithm 2, and we used this modified program to generate 100 alternate versions of s_{self} and s_{EVC} . We then compared these alternates to the original s_{self} and s_{EVC} using stratDiff. In the case of s_{self} , we found that all 100 alternate versions were identical to the original. In the case of s_{EVC} , we found that 58 alternate versions were identical to the original, and the rest exhibited a stratDiff of no more than 1.08×10^{-4} . This means that an agent using an alternate version of s_{EVC} would choose all the same actions as one using the original s_{EVC} at least 99.989% of the time. This tells us that not only does Algorithm 2 converge for the environment we are testing it in, it converges to the same strategy each time it is run. This suggests that the algorithm is finding a globally-best solution for this environment, rather than getting stuck in a local maximum.

Finally, to get an idea of how different s_{self} and s_{EVC} are, we calculated $\text{stratDiff}(s_{\text{self}}, s_{\text{EVC}})$ and found it to be 0.27. This means that training a strategy against an external, fixed strategy in Algorithm 2 does produce significantly different results than training a strategy against itself. For a more in-depth look at where s_{self} and s_{EVC} differ, see Section 9.2.4.

9.2.2 Pairwise Competitions: s_{self} vs. EVChooser and s_{EVC} vs. EVChooser

We played both of our generated strategies, s_{self} and s_{EVC} , against EVChooser for 20,000 games – in 10,000 games, our strategy was defending against an invading population of EVChooser agents, and in 10,000 games the roles were reversed, with our strategy invading and EVChooser defending. We recorded the population of each strategy on every round, as well as the winner of every game.¹¹ The populations in an individual game were extremely noisy, as seen in Figure 3(e), however by averaging the populations over

¹⁰*discountmachine* differs from EVChooser largely because it modifies the expected value of Observing using a machine-learned function that accounts for observe actions being unreliable and returning multiple actions, neither of which are possible in our version of the game

¹¹Recall that the winner of a Cultaptation game is the strategy with the highest average population over the last quarter of the game.

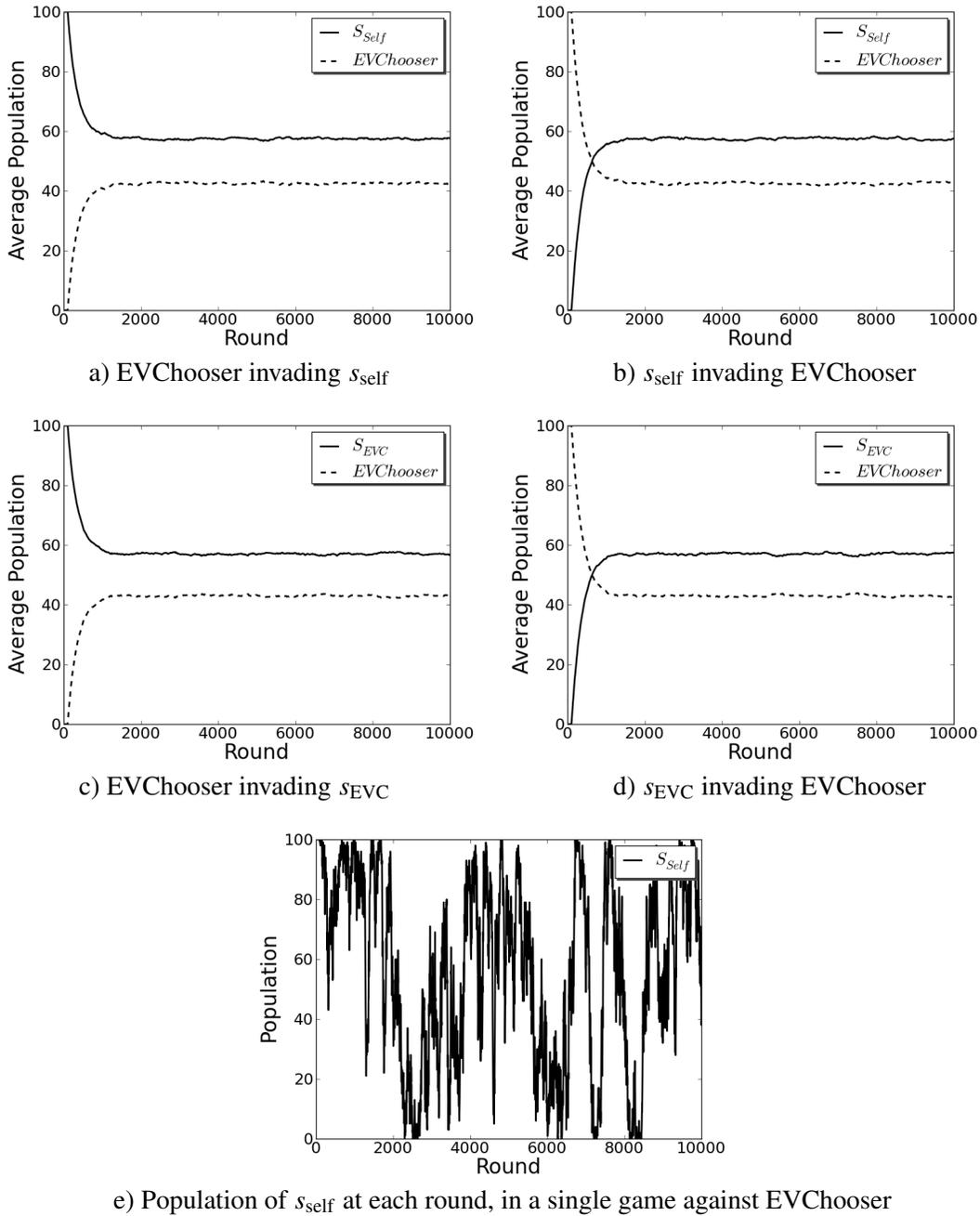


Figure 3: Average populations of both strategies for each round, in match-ups between s_{self} and EVChooser (parts a and b) and between s_{EVC} and EVChooser (parts c and d), over 10,000 games. From round 2000 onwards, s_{self} or s_{EVC} control 57% of the population on average, regardless of whether EVChooser was invading or defending. Since mutation is enabled from round 100 onwards, populations in an individual game (exhibited in part e) are highly mercurial and do not converge. Therefore, we must run a large number of trials and average the results to get a good idea of each strategy's expected performance.

Table 6: Win percentages of s_{self} and s_{EVC} when playing against EVChooser over 10,000 games as both Defender and Invader.

	Win percentage	
	Defending vs. EVChooser	Invading vs. EVChooser
s_{self}	70.65%	70.16%
s_{EVC}	69.92%	69.92%

Table 7: Percentage of games won (out of 10,000) by s_{self} , s_{EVC} , and EVChooser in a melee contest between all three.

	s_{self}	s_{EVC}	EVChooser
Melee win percentage	38.78%	37.38%	23.84%

all 10,000 games we can see some trends emerge. These average populations for each strategy in all four match-ups are presented in Figure 3(a–d), while the win rates for each match-up are presented in Table 6.

In Figure 3 we see that, on average, the strategies generated by Algorithm 2 control roughly 57% of the population for the majority of the game in all four match-ups. Interestingly, both s_{self} and s_{EVC} are able to reach this point in roughly the same amount of time whether they are invading or defending. It is also worth noting that, even though we showed above that s_{self} and s_{EVC} have significant differences, they performed almost identically against EVChooser in terms of population and win percentages

9.2.3 Melee Competition: s_{self} vs. s_{EVC} vs. EVChooser

Our next experiment was to run s_{self} , s_{EVC} , and EVChooser against one another in a melee contest to see how the three strategies would interact in an environment where none of them originally had the upper hand. All three strategies had an initial population of 33 agents at the start of each game. We used the same π_{INV} , probability of change, and probability of death as in Experiment 2. Mutation was disabled for the final 2,500 rounds of each melee game, as was done in the Cultaptation tournament to allow the population to settle. We ran 10,000 games in this manner, and percentage of wins for each strategy are shown in Table 7.

In the table we can see that s_{self} has a slight edge over s_{EVC} , and both these strategies have a significant advantage over EVChooser. In fact, we observed that in the first 100 rounds of most games (before mutation begins) EVChooser nearly died out completely, although it is able to gain a foothold once mutation commences. Mutation is also turned off after 7500 rounds in Cultaptation melee games; this caused the population to quickly become dominated by one of the three strategies in all 10,000 games played.

9.2.4 Performance Analysis of s_{self} , s_{EVC} , and EVChooser

In the experiments in Section 9.2.2, we saw that the strategies found by CSLA consistently outperform EVChooser in environments similar to the ones they were trained in. In order to get a better idea of why this happens, we ran two experiments to compare the performance of s_{self} and EVChooser in more detail. The first was designed to show us the kinds of situations in which the two strategies chose different actions, while the second was designed to let us see how well the two strategies were able to spread good actions through their population.

Action Preferences The objective of this experiment was to identify the kinds of situations in which s_{self} , s_{EVC} , and EVChooser made different choices. To this end, we allowed s_{self} to play against itself for five games, in an environment identical to the one used for the previous experiments in Section 9.2 (note that

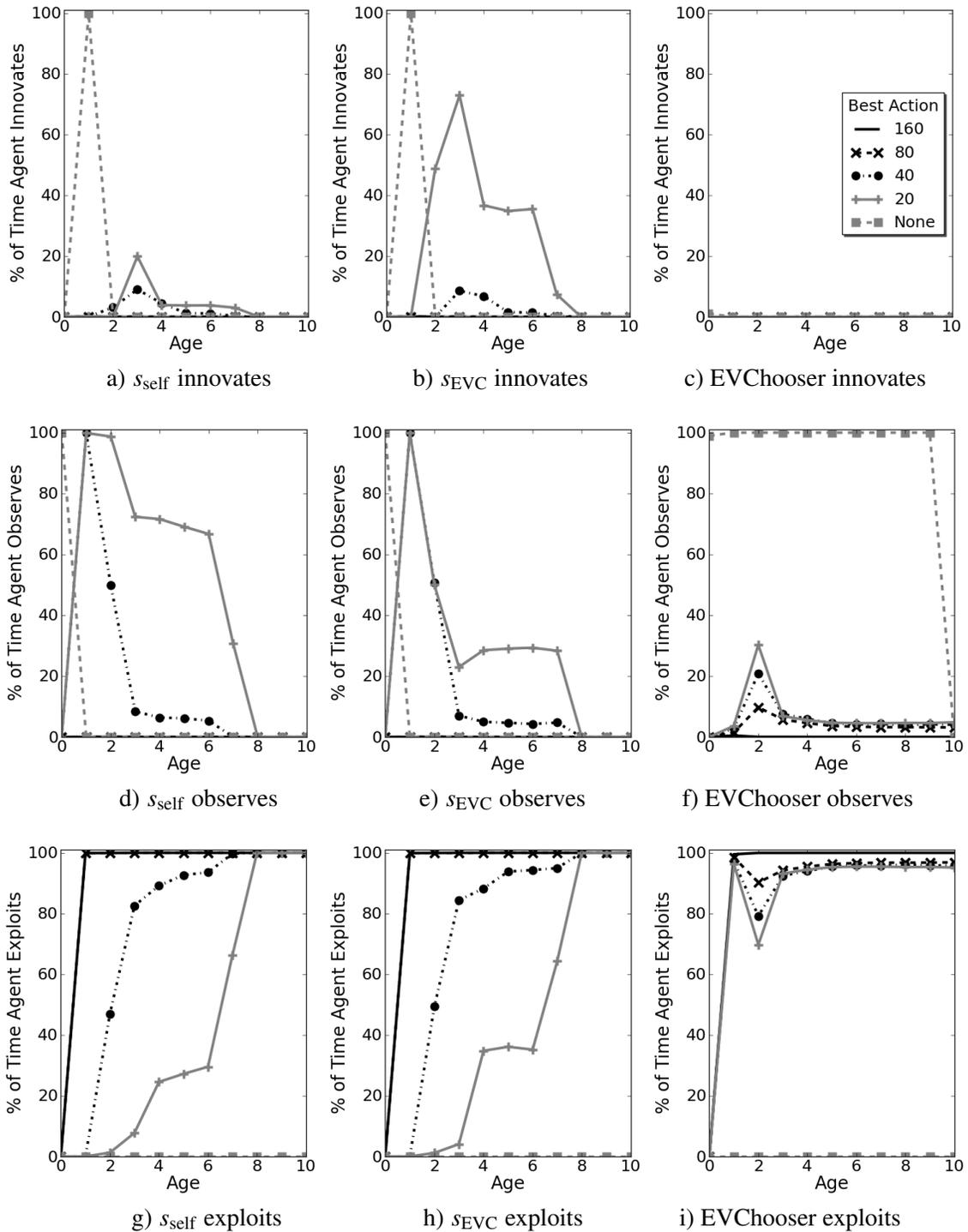


Figure 4: The observed probability that s_{self} , s_{EVC} , and EVChooser will innovate, observe, or exploit when they are a given number of rounds old (on the x -axis) and with a given value of the best action in the agent's repertoire. These results were observed by allowing each strategy to play itself for five games of 10,000 rounds each with 100 agents alive on each round, generating a total of 5,000,000 samples. All graphs in this figure share the same legend, which is included in graph c) and omitted elsewhere to save space.

this is the same environment s_{self} was trained in). On each round, for each agent, we recorded the number of rounds the agent had lived, the value of the best action in its repertoire,¹² and whether the agent chose to innovate, exploit, or observe on that round. Since there are 100 agents alive at any given time and each game lasts ∞ rounds, this gave us five million samples. Figures 4(a), (d), and (g) show the observed probability that s_{self} would innovate, observe, or exploit (respectively) for its first ten rounds and for each possible best action value. We then repeated this process for s_{EVC} and EVChooser, allowing each strategy to play against itself for five games and recording the same data. The results for s_{EVC} and EVChooser may be found in Figures 4(b), (e), and (h), and Figures 4(c), (f), and (i), respectively.

The most obvious difference among the three strategies is that EVChooser almost never innovates,¹³ a property it shares with the strategies that did well in the Cultaptation tournament [34]. On the other hand, s_{self} and s_{EVC} have conditions under which they innovate and conditions under which they do not. For instance, both s_{self} and s_{EVC} always innovate if their first action (which is always an observation) returns no action. Also, s_{EVC} frequently innovates if it is stuck with the worst action after several observes, and s_{self} also innovates (although less frequently; see next paragraph) in this case. Another sharp contrast between EVChooser and the generated strategies is in their exploitation actions. EVChooser spends nearly all of its time exploiting, even if it has a low-value action, and only observes with significant probability on round two. On the other hand, s_{self} and s_{EVC} will begin exploiting immediately if they have one of the two best actions, but otherwise will spend several rounds observing or innovating to attempt to find a better one, and the number of rounds they spend searching for a better action increases as the quality of their best known action decreases.

The main difference between s_{self} and s_{EVC} that can be seen in Figure 4 is in the way they handle being stuck with the lowest-value action after several rounds. In these circumstances, s_{self} prefers observation while s_{EVC} prefers innovation. Here we see the most obvious impact of the differing environments used to generate these two strategies. s_{self} prefers observation in these cases because it was trained in an environment where all agents are willing to perform innovation. Therefore, if an s_{self} agent is stuck with a bad action for more than a few rounds it will continue to observe other agents, since if a better action exists, it is likely that it has already been innovated by another agent and is spreading through the population. On the other hand, s_{EVC} prefers innovation in these situations because it has been trained with EVChooser occupying a significant portion of the population, and we have seen that EVChooser almost never innovates. Therefore, if s_{EVC} is stuck with a bad action after several rounds, it will attempt to innovate to find a better one, since it is less likely that another agent has already done so.

Spreading High-value Actions The objective of this experiment was to measure the rate at which s_{self} , s_{EVC} , and EVChooser were able to spread high-valued actions through their populations. To measure this, we again played s_{self} against itself in the same environment used in the previous experiment (which we will refer to as the “normal” environment in this section), and on each round we recorded the number of agents exploiting actions with each of the four possible values (20, 40, 80, and 160). To account for the noise introduced by changing action values, we ran 10,000 games and averaged the results for each round. We then repeated this process, playing s_{EVC} and EVChooser against themselves. The results for s_{self} , s_{EVC} , and EVChooser may be found in Figures 5(a), (c), and (e) respectively.

This experiment lets us see what the steady state for these strategies looks like, and how quickly they are able to reach it. However, we are also interested in seeing how they respond to structural shocks [10, 14] (i.e. how quickly the strategies are able to recover when a good, widely-used action changes values). To this end, we created a “shock” environment, which is identical to the normal environment with one modification: actions with value 160 have a probability of change equal to 0 except on rounds divisible by 100, in which

¹²This could be 20, 40, 80, 160, or None if the agent had not yet discovered an action

¹³EVChooser innovates 1% of the time on its first round.

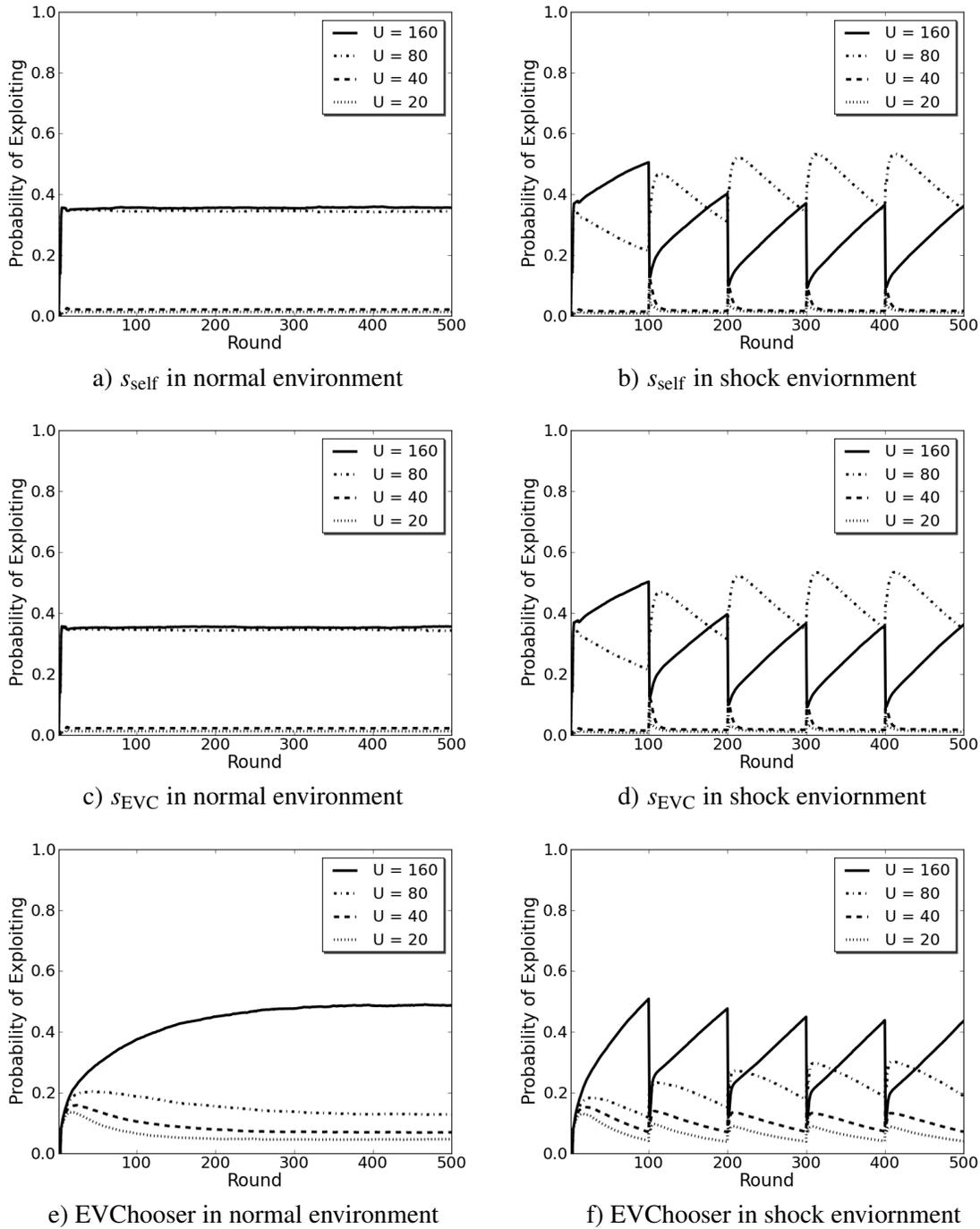


Figure 5: The average number of agents exploiting an action with value U in two environments. The “normal” environment in parts a, c, and e shows how quickly s_{self} , s_{EVC} , and EVChooser spread actions through their population under normal circumstances when they control the entire population. The “shock” environment in parts b, d, and f shows how quickly each strategy responds to periodic structural shock. The “normal” environment is the same as in the rest of Section 9.2, and the “shock” environment is similar except that actions with value 160 are forced to change every 100th round and held constant all other rounds. Each data point is an average over 10,000 games.

case they have probability of change equal to 1. All other actions use the normal probability of change for this environment, 0.01. This modification creates a shock every 100 rounds, while still keeping the expected number of changes the same for all actions. We then repeated the experiment above with the shock environment, running 10,000 games for s_{self} , s_{EVC} , and EVChooser and averaging the results, which are presented in Figures 5(b), (d), and (f) respectively.

In Figure 5 we can see that s_{self} and s_{EVC} exhibit nearly identical performance in both the normal and shock environments. In the normal environment, they are able to reach their steady state in only a few rounds, and the steady state consists of a roughly equal number of agents exploiting the best and second-best action. In the shock environment, we see that s_{self} and s_{EVC} respond to external shock by drastically increasing the number of agents exploiting the second-best action over the course of a few rounds, and returning to their steady states at a roughly linear rate over the next 100 rounds. The number of s_{self} and s_{EVC} agents exploiting the two worst actions remains extremely low except for small spikes immediately after each shock.

Compared to the generated strategies, EVChooser’s performance appears to be less stable, and less robust to structural shock. In the normal environment, we see that EVChooser takes hundreds of rounds to reach its steady state. While EVChooser’s steady state does include more agents exploiting the best action than s_{self} and s_{EVC} , it also includes a significant number of agents exploiting the two worst actions. In the shock environment, we see that changes to the best action result in significant increases to the number of EVChooser agents exploiting the other actions, including the two worst ones. We can also see that populations of EVChooser agents take a lot longer to return to normal after an external shock than populations of s_{self} and s_{EVC} . These results help us account for the superior performance of s_{self} and s_{EVC} over EVChooser in previous experiments, and indicate that there is plenty of room for improvement in EVChooser and strategies like it.

10 Conclusion

In this paper, we have obtained several results that we hope will help provide insight into the utility of inter-agent communication in evolutionary environments. These results can be divided into two main classes: algorithms for computing strategies and equilibria for Cultaptation and similar games, and properties of Cultaptation found by examining the strategies generated by the algorithms.

10.1 Algorithms

Generating near-best-response strategies. In Section 7 we described an algorithm that, for any set $\mathbf{S}_{-\alpha}$ of available strategies other than our own, can construct a nonstationary strategy s_{opt} that is a best response to the other strategies in an infinite Cultaptation game. The algorithm performs a finite-horizon search, and is generalizable to other evolutionary games in which there is a fixed upper bound on per-round utility and a nonzero lower bound on the probability of death at each round. In Section 7.5 we described a state-aggregation technique that speeds up the algorithm by an exponential amount; the state-aggregation technique is generalizable to other evolutionary games in which the utilities are Markovian.

Even with this speedup, computing s_{opt} is not feasible on large instances of Cultaptation, because it requires too much time and space. But the same algorithm can be used to compute, for any $\epsilon > 0$, a much simpler strategy that is an ϵ -best response (i.e., its expected utility is within ϵ of s_{opt} ’s). This computation takes much less time and space.

Computing symmetric near-Nash equilibria. In Section 8 we introduced the Cultaptation Strategy Learning Algorithm (CSLA), which runs the strategy-generation iteratively in order to compute a strategy

s_{self} that is a symmetric near-Nash Equilibrium. Our experiments in Section 9.2 show that CSLA converges to s_{self} in just a few iterations. We have argued that s_{self} is likely to be evolutionarily stable, but have not yet proved this. We hope to prove it in our future work.

10.2 Properties of Cultaptation

The strategies generated by our algorithm show a clear preference for *Observe* actions over *Innovate* actions. The same bias appeared in the Cultaptation tournament, where the best strategies relied almost exclusively on observation. This result was a surprise to tournament organizers Rendell et al. who expected to see innovation performed more frequently among winning strategies [34].

Probably the reason for the tournament organizers surprise was that the existing literature on social learning in evolutionary games has suggested that learning through observation might have only limited value in games like Cultaptation. Reasoning about a slightly simpler model, Rogers concluded that populations would still retain a significant portion of individual learners even if social learning was allowed [35]. Boyd and Richerson extended Rogers’ result, claiming that social learning would not improve the average fitness of a population in any evolutionary scenario, as long as the only benefit to social learning was that it avoided the costs of individual learning [4].

In contrast, we are able to show that s_{self} is an effective strategy despite performing mostly *Observe* actions. We can also provide specific details about when and in what conditions *Innovate* becomes preferable. Rather than relying exclusively on *Observe* actions, s_{self} will innovate if an agent is stuck with a low-valued action after several attempted observations. This effect is even more prominent in s_{EVC} , which was far more likely than s_{self} to perform innovate actions if the first few observations failed to produce a good value. Figure 4 shows a noticeable increase in the number of *Innovate* actions performed by s_{EVC} when the agent still has a low value by the third round of its life. Based on the data in Figure 5, we know that the quality of π_{Obs} is not as good in an environment with *EVChooser*, so s_{EVC} will resort to innovation both more quickly and more often than s_{self} .

Rendell et al. attribute the effectiveness of *Observe* actions in the tournament to the “filtering” process performed by other agents. Since most agents will act rationally by exploiting their best action, an *Observe* action will quickly reveal high valued actions to the observer. Our results show a similar phenomenon, where s_{self} is able to propagate high-valued actions more quickly not simply because it exploits its best action, but also because it hesitates to exploit low-valued actions and spends that time learning instead. By not polluting π_{Obs} with low-valued exploits, the probability that a high-valued action will be observed actually increases. The consequence of this is shown in Figure 5, where high-valued actions propagate very quickly in a population of s_{self} agents, and more slowly in a population of *EVChooser*. The graph in Figure 4 shows that *EVChooser* is far more likely than s_{self} to exploit low-valued actions.

10.3 Future Work

One limitation of this work is that we were unable to compare s_{self} , the strategy generated by our CSLA, against the best-performing agents in the Cultaptation tournament. *EVChooser*, the strategy that we used as an invader in Section 9.2, placed 15th in the Cultaptation tournament. In the future, we hope to get the source code for the top-performing agents and test our algorithms against them.

Also left for future work is the examination of information-gathering in the social learning game. An agent in the game would not normally be able to compute a best-response strategy as we have done in this paper, because it would not know the other players’ strategies, nor the probability distributions from which the innovation and observation utilities are drawn. Such an agent would need either to approximate the distributions or to use an algorithm that can do well without them. If we choose to approximate, should our agent be willing to sacrifice some utility early on, in order to gain information that will improve its

approximation? Are there strategies that perform well in a wide variety of environments, that we could use until our agent develops a good approximation? Are some of these strategies so versatile that we can simply use them without needing to know the distributions? These remain open questions.

11 Acknowledgements

This work was supported in part by AFOSR grants FA95500610405 and FA95501210021, and Army Research Office cooperative agreement W911NF0920072. The opinions in this paper are those of the authors and not necessarily those of the funders.

References

- [1] A. Bandura. *Social Learning Theory*. General Learning Press, New York, 1977.
- [2] C.J. Barnard and R. M. Sibly. Producers and scroungers: A general model and its application to captive flocks of house sparrows. *Animal Behavior*, 29:543–550, 1981.
- [3] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *International Joint Conference on Artificial Intelligence*, volume 14, pages 1104–1113. LAWRENCE ERLBAUM ASSOCIATES LTD, 1995.
- [4] R. Boyd and P.J. Richerson. Why does culture increase human adaptability? *Ethology and Sociobiology*, 16(2):125–143, 1995.
- [5] Robert Boyd, Magnus Enquist, Kimmo Eriksson, Marcus Feldman, and Kevin Laland. Cultaptation: Social learning tournament, 2008. <http://www.intercult.su.se/cultaptation>.
- [6] Ryan Carr, Eric Raboin, Austin Parker, and Dana Nau. When innovation matters: An analysis of innovation in a social learning game. In *Second International Conference on Computational Cultural Dynamics (ICCCD)*, September 2008.
- [7] Michael Childers. personal communication.
- [8] Grant A. Duffy, Thomas W. Pike, and Kevin N. Laland. Size-dependent directed social learning in nine-spined sticklebacks. *Animal Behaviour*, 78(2):371–375, August 2009.
- [9] M. Enquist, S. Ghirlanda, and K. Eriksson. Critical social learning: A solution to rogers’s paradox of nonadaptive culture. *American Anthropologist*, 109(4):727–734, 2007.
- [10] D. Friedman. On economic applications of evolutionary game theory. *Journal of Evolutionary Economics*, 8(1):15–43, 1998.
- [11] B.G. Galef and K.N. Laland. Social learning in animals: Empirical studies and theoretical models. *Bioscience*, 55:489–499, 2005.
- [12] L. A. Giraldeau, T. J. Valone, and J. J. Templeton. Potential disadvantages of using socially acquired information. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 357(1427):1559–1566, 2002.
- [13] Sudipto Guha, Kamesh Munagala, and Peng Shi. Approximation algorithms for restless bandit problems. In *SODA '09: Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 28–37, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

- [14] Eran Guse. Expectational business cycles. Money Macro and Finance (MMF) Research Group Conference 2004 97, Money Macro and Finance Research Group, September 2004.
- [15] J. Henrich and R. McElreath. The evolution of cultural evolution. *Evolutionary Anthropology*, 12:123–135, 2003.
- [16] J. Hofbauer and K. Sigmund. Evolutionary game dynamics. *Bulletin of the American Mathematical Society*, 40(4):479, 2003.
- [17] E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, Potomac, MD, 1978.
- [18] R.A. Howard. *Dynamic programming and Markov process*. MIT Press, 1960.
- [19] T. Ibaraki. Theoretical comparison of search strategies in branch and bound. *International Journal of Computer and Information Sciences*, 5:315–344, 1976.
- [20] Tatsuya Kameda and Daisuke Nakanishi. Cost–benefit analysis of social/cultural learning in a nonstationary uncertain environment: An evolutionary simulation and an experiment with human subjects. *Evolution and Human Behavior*, 23:373–393, September 2002.
- [21] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes. *MACHINE LEARNING*, 49:193–208, 2002.
- [22] Jeremy R. Kendal, Luke Rendell, Thomas W. Pike, and Kevin N. Laland. Nine-spined sticklebacks deploy a hill-climbing social learning strategy. *Behavioral Ecology*, 20(2):238–244, 2009.
- [23] D. Koller and R. Parr. Computing factored value functions for policies in structured MDPs. In *International Joint Conference on Artificial Intelligence*, volume 16, pages 1332–1339. Citeseer, 1999.
- [24] D.E. Koulouriotis and A. Xanthopoulos. Reinforcement learning and evolutionary algorithms for nonstationary multi-armed bandit problems. *Applied Mathematics and Computation*, 196(2):913 – 922, 2008.
- [25] K.N. Laland. Social learning strategies. *Learning and Behavior*, 32:4–14, 2004.
- [26] Kevin Leyton-Brown and Yoav Shoham. *Essentials of Game Theory: A Concise Multidisciplinary Introduction*. Morgan & Claypool, 2008.
- [27] R. McElreath, M. Lubell, P. Richerson, T. Waring, W. Baum, E. Edsten, C. Efferson, and B. Paciotti. Applying evolutionary models to the laboratory study of social learning. *Evolution and Human Behavior*, 26(6):483–508, November 2005.
- [28] D. Nettle. Language: Costs and benefits of a specialised system for social information transmission. In J. Wells and et al., editors, *Social Information Transmission and Human Biology*, pages 137–152. Taylor and Francis, London, 2006.
- [29] J Noble, P M Todd, and E Tuci. Explaining social learning of food preferences without aversions: an evolutionary simulation model of norway rats. *Proc. Biol Sci.*, 268(1463):141–149, January 2001.
- [30] Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of optimal queuing network control. *Mathematics of Operations Research*, 24(2):293–305, May 1999.

- [31] Aske Plaat, Jonathan Schaeffer, Wim Pijls, and Arie de Bruin. An algorithm faster than negascout and sss* in practice. In *Computer Strategy Game Programming Workshop*. 1995.
- [32] Eric Raboin, Ryan Carr, Austin Parker, and Dana Nau. Balancing innovation and exploitation in a social learning game. In *AAAI Fall Symposium on Adaptive Agents in Cultural Contexts*, November 2008.
- [33] L. G. Rapaport and G. R. Brown. Social influences on foraging behavior in young nonhuman primates: Learning what, where, and how to eat. *Evolutionary Anthropology*, 17(4):189–201, 2008.
- [34] L. Rendell, R. Boyd, D. Cownden, M. Enquist, K. Eriksson, M. W. Feldman, L. Fogarty, S. Ghirlanda, T. Lillicrap, and K. N. Laland. Why copy others? insights from the social learning strategies tournament. *Science*, 328(5975):208–213, April 2010.
- [35] A. R. Rogers. Does biology constrain culture? *American Anthropologist*, 90(4):819–831, 1988.
- [36] K.H. Schlag. Why imitate, and if so, how?, : A boundedly rational approach to multi-armed bandits. *Journal of Economic Theory*, 78:130–156, 1998.
- [37] John Carl Villaneuva. Atoms in the universe. *Universe Today*, July 2009. <http://www.universetoday.com/36302/atoms-in-the-universe/>.
- [38] Thomas N. Wisdom and Robert L. Goldstone. Social learning and cumulative innovations in a networked group. In *Advances in Social Computing: Third International Conference on Social Computing, Behavioral Modeling, and Prediction, SBP 2010*, pages 32–41, 2010.
- [39] T.R. Zentall. Imitation: Definitions, evidence, and mechanisms. *Animal Cognition*, 9:335–353, 2006.

A Proofs

Proposition 2 $\text{EPRU}(s \mid G, \mathbf{S})/d = \text{EPRU}_{\text{alt}}(s, \langle \rangle \mid G, \mathbf{S})$.

Proof.

First, we will show by induction that $\text{EPRU}_{\text{alt}}(s, \langle \rangle \mid G, \mathbf{S})$ equals the summation of $P(h_\alpha | s_\alpha, \mathbf{S}) \text{EV}_{\text{exp}}(|h_\alpha|, U(h_\alpha[|h_\alpha|]))$ for all histories h_α . Then we will show that this equals the summation of $L(|h_\alpha|)P(h_\alpha | s_\alpha, \mathbf{S}) \text{PRU}(h_\alpha)$ for all h_α .

We will begin with the definition of EPRU_{alt} in Equation 13, and note that $P(\langle \rangle \circ t | \langle \rangle, s_\alpha(\langle \rangle), \mathbf{S}) = P(\langle \rangle \circ t | s_\alpha, \mathbf{S})$ for histories of length one. This gives us a base case of

$$\text{EPRU}_{\text{alt}}(s_\alpha, \langle \rangle \mid G, \mathbf{S}) = \sum_{t \in T} P(\langle \rangle \circ t | s_\alpha, \mathbf{S}) \text{EV}_{\text{exp}}(1, U(t)) + \sum_{t \in T} P(\langle \rangle \circ t | s_\alpha, \mathbf{S}) \text{EPRU}_{\text{alt}}(s_\alpha, \langle \rangle \circ t \mid G, \mathbf{S}).$$

For the inductive case, we will again start from Equation 13, this time noting that $P(h_\alpha | s_\alpha, \mathbf{S}) P(h_\alpha \circ t | h_\alpha, s_\alpha(h_\alpha), \mathbf{S})$ simplifies to just $P(h_\alpha \circ t | s_\alpha, \mathbf{S})$. Thus, for all h_α we can rewrite $P(h_\alpha | s_\alpha, \mathbf{S}) \text{EPRU}_{\text{alt}}(s_\alpha, h_\alpha \mid G, \mathbf{S})$ in terms of histories one round longer than h_α , as follows:

$$\begin{aligned} P(h_\alpha | s_\alpha, \mathbf{S}) \text{EPRU}_{\text{alt}}(s_\alpha, h_\alpha \mid G, \mathbf{S}) &= \sum_{t \in T} P(h_\alpha \circ t | s_\alpha, \mathbf{S}) \text{EV}_{\text{exp}}(|h_\alpha \circ t|, U(t)) \\ &\quad + \sum_{t \in T} P(h_\alpha \circ t | s_\alpha, \mathbf{S}) \text{EPRU}_{\text{alt}}(s_\alpha, h_\alpha \circ t \mid G, \mathbf{S}). \end{aligned}$$

Therefore, by induction we have

$$\text{EPRU}_{\text{alt}}(s_\alpha, \langle \rangle \mid G, \mathbf{S}) = \sum_{h_\alpha \in H} P(h_\alpha | s_\alpha, \mathbf{S}) \text{EV}_{\text{exp}}(|h_\alpha|, U(h_\alpha[|h_\alpha|])),$$

where H is the set of all possible histories.

The proof then proceeds arithmetically:

$$\begin{aligned} \text{EPRU}_{\text{alt}}(s_\alpha, \langle \rangle \mid G, \mathbf{S}) &= \sum_{h_\alpha \in H} P(h_\alpha | s_\alpha, \mathbf{S}) \text{EV}_{\text{exp}}(|h_\alpha|, U(h_\alpha[|h_\alpha|])) \\ &= \sum_{h_\alpha \in H} P(h_\alpha | s_\alpha, \mathbf{S}) \sum_{i=|h_\alpha|}^{\infty} \frac{L(i)U(h_\alpha[|h_\alpha|])}{i} \\ &= \sum_{h_\alpha \in H} \sum_{i=|h_\alpha|}^{\infty} \frac{L(i)P(h_\alpha | s_\alpha, \mathbf{S})U(h_\alpha[|h_\alpha|])}{i} \\ &= \sum_{i=1}^{\infty} \sum_{h_\alpha \in H(\leq i)} \frac{L(i)P(h_\alpha | s_\alpha, \mathbf{S})U(h_\alpha[|h_\alpha|])}{i} \\ &= \sum_{i=1}^{\infty} \sum_{h_\alpha \in H(i)} L(i)P(h_\alpha | s_\alpha, \mathbf{S}) \text{PRU}(h_\alpha) = \text{EPRU}(s_\alpha \mid G, \mathbf{S})/d \end{aligned}$$

Where $H(\leq i)$ is the set of all histories of length less than or equal to i , and $H(i)$ is the set of all histories exactly of length i . \square

Proposition 3 $\text{Strat}(h_\alpha, k, V, \mathbf{S})$ returns (s_α, U) such that

$$\text{EPRU}_{\text{alt}}^k(s_\alpha, h_\alpha \mid G, \mathbf{S}) = U = \max_{s'} (\text{EPRU}_{\text{alt}}^k(s', h_\alpha \mid G, \mathbf{S})).$$

Proof. Let s' be a strategy maximizing $\text{EPRU}_{\text{alt}}^k(s', h_\alpha | G, \mathbf{S})$ and let $\{s_\alpha, U\}$ be the strategy and value returned by $\text{Strat}(h_\alpha, k, V, \mathbf{S})$. We will show by induction on k that

$$\text{EPRU}_{\text{alt}}^k(s_\alpha, h_\alpha | G, \mathbf{S}) = U = \text{EPRU}_{\text{alt}}^k(s', h_\alpha | G, \mathbf{S}).$$

In the base case, $k = 0$ and clearly $\text{EPRU}_{\text{alt}}^0(s', h_\alpha | G, \mathbf{S}) = 0$ for any s' , therefore $\text{EPRU}_{\text{alt}}^k(s_\alpha, h_\alpha | G, \mathbf{S}) = \text{EPRU}_{\text{alt}}^k(s', h_\alpha | G, \mathbf{S}) = 0 = U$ as required.

For the inductive case, suppose that for k , $\text{Strat}(h_\alpha, k, V, \mathbf{S})$ returns $\{s_\alpha, U\}$ such that $\text{EPRU}_{\text{alt}}^k(s_\alpha, h_\alpha | G, \mathbf{S}) = U = \max_{s'}(\text{EPRU}_{\text{alt}}^k(s', h_\alpha | G, \mathbf{S}))$. We must then show that $\text{Strat}(h_\alpha, k + 1, V, \mathbf{S})$ returns $\{s_\alpha, U\}$ such that

$$\text{EPRU}_{\text{alt}}^{k+1}(s_\alpha, h_\alpha | G, \mathbf{S}) = U = \max_{s'}(\text{EPRU}_{\text{alt}}^{k+1}(s', h_\alpha | G, \mathbf{S})).$$

Let s_{temp} be the strategy constructed in lines 8–19 of the algorithm. First we show that on line 20,

$$\begin{aligned} & \text{EPRU}_{\text{alt}}^{k+1}(s_{\text{temp}}, h_\alpha | G, \mathbf{S}) = \\ & \sum_{t \in T} P(h_\alpha \circ t | h_\alpha, s_{\text{temp}}(h_\alpha), \mathbf{S}) \left(\text{EV}_{\text{exp}}(|h_\alpha|, U(t)) + \text{EPRU}_{\text{alt}}^k(s_{\text{temp}}, h_\alpha \circ t | G, \mathbf{S}) \right) = U_{\text{temp}}. \end{aligned} \quad (20)$$

This follows because the t on line 11 iterates over all possible $t \in T$ (due to the `for` loops on lines 6, 9, and 10), meaning that the eventual value of U_{temp} is

$$\sum_{t \in T} P(h_\alpha \circ t | h_\alpha, s_{\text{temp}}(h_\alpha) | \mathbf{S}) \left(\text{EV}_{\text{exp}}(|h_\alpha \circ t|, U(t)) + U' \right).$$

By the inductive hypothesis, $U' = \text{EPRU}_{\text{alt}}^k(s_{\text{temp}}, h_\alpha \circ t | G, \mathbf{S})$, sufficing to show that (20) holds.

Now we show that

$$\text{EPRU}_{\text{alt}}^{k+1}(s_\alpha, h_\alpha | G, \mathbf{S}) = \text{EPRU}_{\text{alt}}^{k+1}(s', h_\alpha | G, \mathbf{S}).$$

Clearly $\text{EPRU}_{\text{alt}}^{k+1}(s_\alpha, h_\alpha | G, \mathbf{S}) \leq \text{EPRU}_{\text{alt}}^{k+1}(s', h_\alpha | G, \mathbf{S})$, since s' is assumed to have maximal $\text{EPRU}_{\text{alt}}^{k+1}$ for h_α , so it suffices to show that

$$\text{EPRU}_{\text{alt}}^{k+1}(s_\alpha, h_\alpha | G, \mathbf{S}) \geq \text{EPRU}_{\text{alt}}^{k+1}(s', h_\alpha | G, \mathbf{S}).$$

Since s_α maximizes

$$\sum_{t \in T} P(h_\alpha \circ t | h_\alpha, s_\alpha(h_\alpha), \mathbf{S}) (\text{EV}_{\text{exp}}(|h_\alpha \circ t|, U(t)) + U'),$$

where $U' \geq \text{EPRU}_{\text{alt}}^k(s', h_\alpha \circ t | G, \mathbf{S})$ by the inductive hypothesis, there can be no action a such that

$$\begin{aligned} & \sum_{t \in T} P(h_\alpha \circ t | h_\alpha, a, \mathbf{S}) (\text{EV}_{\text{exp}}(|h_\alpha \circ t|, U(t)) + \text{EPRU}_{\text{alt}}^k(s', h_\alpha \circ t | G, \mathbf{S})) \\ & > \sum_{t \in T} P(h_\alpha \circ t | h_\alpha, s'(h_\alpha), \mathbf{S}) (\text{EV}_{\text{exp}}(|h_\alpha \circ t|, U(t)) + U'). \end{aligned}$$

Therefore $\text{EPRU}_{\text{alt}}^{k+1}(s_\alpha, h_\alpha | G, \mathbf{S}) \geq \text{EPRU}_{\text{alt}}^{k+1}(s', h_\alpha | G, \mathbf{S})$. This concludes the inductive argument.

Thus for all k , $\text{EPRU}_{\text{alt}}^k(s_\alpha, h_\alpha | G, \mathbf{S}) = U = \max_{s'} \text{EPRU}_{\text{alt}}^k(s', h_\alpha | G, \mathbf{S})$. \square

Algorithm 3 Creates and returns a repertoire for history h_α .

CreateRepertoire($h_\alpha = (a_1, (m_1, v_1)), \dots, (a_r, (m_r, v_r))$)

Let $\mu = \{m_i | i = 1, \dots, r\}$.

Let $R = \emptyset$ { R will be the repertoire.}

for $m \in \mu$ **do**

Let $i = \max_{\{1, \dots, r\}} (m_i = m)$

Add $\langle v_i, r - i \rangle$ to R .

end for

return R

B Converting from Histories to Repertoires

In this appendix, we will formally define a repertoire, explain how to transform histories into repertoires, and show that the number of possible repertoires is substantially smaller than the number of possible histories, while maintaining the property that any best-response action for a given repertoire is also a best response for any history associated with that repertoire (Theorem 5). Finally, we will present a modified version of Algorithm 1, which uses repertoires rather than histories, and we will show how this simple change cuts the branching factor of the algorithm in half (Algorithm 4).

B.0.1 Repertoire Definition

A repertoire tells the last value and *age* of each action an agent “knows,” where an action’s age is the number of rounds that have passed since the agent last obtained information about it. Since at any given point in a game, each known action has a unique age, we label exploitation actions by their value and age, leaving off the action number (e.g. if we discovered an action with value 4 last round and an action with value 26 three rounds ago, then the repertoire will be $\{\langle 4, 1 \rangle, \langle 26, 3 \rangle\}$ where $\langle 4, 1 \rangle$ denotes the existence of an action with value 4 discovered 1 round ago, and $\langle 26, 3 \rangle$ denotes the existence of an action with value 26 discovered 3 rounds ago). Formally, a repertoire is defined to be a set of pairs, where the first value in each pair represents the knowledge of an action with the given value, while the second value in the pair represents the number of rounds since that knowledge was last updated.

Definition. Let $v_1, \dots, v_m \in V$ be action values and $\gamma_1, \dots, \gamma_m \in \mathbb{Z}^+$ (the positive integers) be action ages. A repertoire R is a set of action value/action age pairs $R = \{\langle v_1, \gamma_1 \rangle, \dots, \langle v_m, \gamma_m \rangle\}$. We denote the set of all repertoires as \mathcal{Rep} , and the set of all repertoires where all $\gamma_i \leq j$ as \mathcal{Rep}_j . \square

\mathcal{Rep} has unbounded size, but \mathcal{Rep}_j has finite size. We show how to create a repertoire R from a history h_α using the CreateRepertoire function in Algorithm 3.

Repertoires change based on the action performed. For example, repertoire

$$R = \{\langle 4, 1 \rangle, \langle 26, 3 \rangle\}$$

can change to repertoire

$$R' = \{\langle 4, 2 \rangle, \langle 26, 4 \rangle, \langle 27, 1 \rangle\}$$

after an innovation action where an action with value 27 is innovated. Notice that all actions in R' , apart from the newly-innovated action with age 1, are one round older than they were in R . This aging process occurs often enough for us to introduce a function which ages a repertoire $R = \{\langle v_i, \gamma_i \rangle\}$:

$$\text{age}(\{\langle v_i, \gamma_i \rangle\}) = \{\langle v_i, \gamma_i + 1 \rangle\}$$

Finally, we will introduce two functions to represent the two ways our repertoire can change when we perform an action.

The first, `newaction`, returns a repertoire with a new action added to it:

$$\text{newaction}(R, v) = \text{age}(R) \cup \{\langle v, 1 \rangle\}$$

The second, `updaction`, returns a repertoire with updated information on action m :

$$\text{updaction}(R, v, m) = \text{age}(R \setminus \{\langle v_m, \gamma_m \rangle\}) \cup \{\langle v, 1 \rangle\}$$

B.0.2 Transition Probabilities

We can now define the probability of transitioning between repertoires on round r . We will call the transition probability functions $P^{\text{Rep}}(R'|R, r, a, \mathbf{S})$ for $a \in \{\text{Inv}, \text{Obs}, \text{X}_i\}$. In general, these functions will mirror the $P(h'|h, a, \mathbf{S})$ functions defined in Section 5, with some extra clauses added to ensure that if it is not possible to go from repertoire R to repertoire R' using the given action, then the transition probability is 0.

Innovation actions For innovation actions, the function is:

$$P^{\text{Rep}}(R'|R, r, \text{Inv}, \mathbf{S}) = \begin{cases} 0 & \text{if } |R| = \mu \vee \nexists v : R' = \text{newaction}(R, v) \\ \pi_{\text{Inv}}(v|r) & \text{if } R' = \text{newaction}(R, v) \end{cases}$$

The first clause ensures that if all possible actions are already in R , or if it is not possible to go from R to R' in one innovation action, then the transition probability is 0. The second clause simply tells us the probability of innovating an action with value v , given that it is possible to go from R to R' in one innovation action.

Observation actions In Section 5 we assumed the existence of a distribution π_{Obs} that, when given the current history, would tell us the probability of observing an action with a given value. Here, we will make the following assumptions about π_{Obs} :

- When given a repertoire and round number, $\pi_{\text{Obs}}(v|R, r, \mathbf{S})$ tells us the probability of observing an action that has value v and is not already known by R .
- When given a repertoire and round number, $\pi_{\text{Obs}}(m, v|R, r, \mathbf{S})$ tells us the probability of observing an action that has value v , and was previously in R at position m . The value of this action may have changed.
- π_{Obs} can make its predictions without using any information lost when converting from a history to a repertoire.

These assumptions are all satisfied by the π_{Obs} used in our implementation, and we expect them to hold for other practical implementations as well, since a distribution conditioned on entire histories would be impractically large.

With this in mind, the transition probability function for observation actions is

$$P^{\text{Rep}}(R'|R, r, \text{Obs}, \mathbf{S}) = \begin{cases} \pi_{\text{Obs}}(m, v|R, r, \mathbf{S}) & \text{if } \langle v'_m, \gamma_m \rangle \in R \wedge \\ & R' = \text{updaction}(R, v, m) \\ \pi_{\text{Obs}}(v|R, r, \mathbf{S}) & \text{if } R' = \text{newaction}(R, v) \\ 0 & \text{Otherwise.} \end{cases}$$

The first clause gives us the probability of observing an action already in our repertoire, while the second gives us the probability of observing a new action.

Exploitation actions Let $\langle v_i, \gamma_i \rangle$ be the value and age of exploitation action X_i . Then

$$P^{Rep}(R'|R, r, X_i, \mathbf{S}) = \begin{cases} 0 & \text{if } |R| \neq |R'| \\ 0 & \text{if } \forall v' \in V, R' \neq \text{updaction}(R, v', i) \\ \prod_{j=r-\gamma_i}^r (1 - c(j)) + \sum_{j=r-\gamma_i}^r c(j) \pi(v_i, j) \left(\prod_{i=j}^r (1 - c(i)) \right) & \text{if } R' = \text{updaction}(R, v_i, i) \\ \sum_{j=r-\gamma_i}^r c(j) \pi(v'_i, j) \left(\prod_{i=j}^r (1 - c(i)) \right) & \text{if } R' = \text{updaction}(R, v'_i, i) \text{ and } v_i \neq v'_i \end{cases}$$

The first two clauses check that we can, in fact, transition between R and R' by exploiting. The third clause gives us the probability that the action we exploited has not changed since we last saw it, while the fourth clause gives us the probability that the action we exploited has changed.

B.0.3 Consistency between P and P^{Rep}

Later in this section, we will show that using a repertoire-based algorithm to compute ϵ -best-response strategies returns the same results as using the history-based Algorithm 1. To do this, we will use the notion of *consistency* between the P and P^{Rep} equations.

Definition. Let M be the set of actions known to an agent with history h_α . The P and P^{Rep} equations are *consistent* for h_α if, for all $a \in \{\text{Inv}, \text{Obs}, X_i\}$ and $v \in V$:

$$\sum_{m \in M} P(h \circ (a, (m, v)) | h, a, \mathbf{S}) = \sum_{m=1}^{|R|} P^{Rep}(\text{updaction}(R, v, m) | R, r, a, \mathbf{S}) \quad (21)$$

and

$$\sum_{m \in \{1, \dots, \mu\} \setminus M} P(h \circ (a, (m, v)) | h, a, \mathbf{S}) = P^{Rep}(\text{newaction}(R, v) | R, r, a, \mathbf{S}) \quad (22)$$

where $R = \text{CreateRepertoire}(h)$ and $r = |h_\alpha|$. □

Lemma 6 *The P and P^{Rep} equations are consistent for all $h \in H$.*

Proof. We can prove this by using the definition of P , found in Section 5, and the definition of P^{Rep} found above. We will simply consider what happens for arbitrary h_α and v when performing innovation, observation, and exploitation actions.

Recall that $X(h)$ returns the number of exploit moves available to an agent with history h . For ease of exposition, we will assume without loss of generality that the first action learned by h_α has label 1, the second has label 2, etc. Thus $M = \{1, \dots, X(h)\}$, while $\{1, \dots, \mu\} \setminus M = \{X(h) + 1, \dots, \mu\}$.

Innovation actions An innovation action always returns information on a new action, so both sides of Equation 21 are clearly 0 in this case. If $X(h) = |R| = \mu$, both sides of Equation 22 are also 0 since no new actions can be innovated. Thus, we will assume $X(h) = |R| < \mu$. We now have

$$\sum_{m=X(h)+1}^{\mu} P(h \circ (\text{Inv}, (m, v)) | h, \text{Inv}, \mathbf{S}) = (\mu - X(h)) \frac{\pi_{\text{Inv}}(v|r)}{\mu - X(h)} = \pi_{\text{Inv}}(v|r)$$

and

$$P^{Rep}(\text{newaction}(R, v) | R, r, \text{Inv}, \mathbf{S}) = \pi_{\text{Inv}}(v|r)$$

which are clearly equivalent. Hence, P and P^{Rep} are consistent on h_α when $a = \text{Inv}$.

Observation actions This section of the proof is mostly trivial, given the assumptions we have made about π_{Obs} . The left side of Equation 21 is

$$\sum_{m=1}^{X(h)} P(h \circ (\text{Obs}, (m, v)) | h, \text{Obs}, \mathbf{S}) = \sum_{m=1}^{X(h)} \pi_{\text{Obs}}(m, v | h, \mathbf{S})$$

which tells us the probability of observing one of the actions already seen in our current history. The right side is

$$\sum_{m=1}^{|\mathcal{R}|} P^{\text{Rep}}(\text{updaction}(R, v, m) | R, r, \text{Obs}, \mathbf{S}) = \sum_{m=1}^{|\mathcal{R}|} \pi_{\text{Obs}}(m, v | R, r, \mathbf{S})$$

Since we assume that $\pi_{\text{Obs}}(m, v | R, r, \mathbf{S})$ tells us the probability of observing action m in the repertoire, $\sum_{m=1}^{|\mathcal{R}|} \pi_{\text{Obs}}(m, v | R, r, \mathbf{S})$ also tells us the probability of observing any of the actions we have already seen. Therefore, Equation 21 holds for observation actions.

For Equation 22, the left side is

$$\sum_{m=X(h)+1}^{\mu} P(h \circ (\text{Obs}, (m, v)) | h, \text{Obs}, \mathbf{S}) = \sum_{m=X(h)+1}^{\mu} \pi_{\text{Obs}}(m, v | h, \mathbf{S})$$

which tells us the probability of observing an action we have not yet seen. The right side is

$$P^{\text{Rep}}(\text{newaction}(R, v) | R, r, \text{Obs}, \mathbf{S}) = \pi_{\text{Obs}}(v | R, r, \mathbf{S})$$

Since we assume that $\pi_{\text{Obs}}(v | R, r, \mathbf{S})$ gives us the probability of observing a new action, Equation 22 also holds for observation actions. Hence, P and P^{Rep} are consistent on h_α when $a = \text{Obs}$.

Exploitation actions Exploiting an action never gives us information about a new action, so both sides of Equation 22 are 0 when we exploit. Thus, we need only consider Equation 21.

We will consider two cases. In the first case, the action we choose to exploit has changed since we last saw it, so v is a new value. We then have

$$\sum_{m=1}^{X(h)} P(h \circ (\mathbf{X}_m, (m, v)) | h, \mathbf{X}_m, \mathbf{S}) = \sum_{m=1}^{X(h)} \sum_{j=\text{last}_m}^r c(j) \pi(v, j) \prod_{i=j}^r (1 - c(i))$$

where last_m is the last round number on which we obtained any information about the m -th action.

Similarly, since exploiting never increases the size of a repertoire, we have

$$\sum_{m=1}^{|\mathcal{R}|} P^{\text{Rep}}(\text{updaction}(R, v, m) | R, r, \mathbf{X}_m, \mathbf{S}) = \sum_{m=1}^{|\mathcal{R}|} \sum_{j=r-\gamma_m}^r c(j) \pi(v, j) \prod_{i=j}^r (1 - c(i))$$

Since $|\mathcal{R}| = X(h)$ and $\text{last}_m = r - \gamma_m$ by definition, Equation 21 holds when the action we exploit changes.

Next, we consider the case where the action we choose to exploit has *not* changed. In this case, we have

$$\sum_{m=1}^{X(h)} P(h \circ (\mathbf{X}_m, (m, v)) | h, \mathbf{X}_m, \mathbf{S}) = \sum_{m=1}^{X(h)} \left(\prod_{j=\text{last}_m}^r (1 - c(j)) + \sum_{j=\text{last}_m}^r c(j) \pi(v, j) \prod_{i=j}^r (1 - c(i)) \right)$$

and

$$\sum_{m=1}^{|\mathcal{R}|} P^{\text{Rep}}(\text{updaction}(R, v, m) | R, r, \mathbf{X}_m, \mathbf{S}) = \sum_{m=1}^{|\mathcal{R}|} \left(\prod_{j=r-\gamma_m}^r (1 - c(j)) + \sum_{j=r-\gamma_m}^r c(j) \pi(v, j) \prod_{i=j}^r (1 - c(i)) \right)$$

which are also equivalent. Therefore, Equation 21 also holds when the action we exploit does not change.

We have now shown that P and P^{Rep} are consistent on arbitrary h_α when $a \in \{\text{Inv}, \text{Obs}, X_i\}$ and for arbitrary v . Therefore, P and P^{Rep} are consistent for all h_α . \square

B.0.4 Repertoire-based Strategies

Repertoires can be used to more compactly define a strategy. We let a *repertoire-based strategy* s be a function from repertoires to actions. Such a strategy can be represented more compactly than the history-based strategies used earlier in this paper, since there are fewer possible repertoires than there are possible histories. In any history h_α , a repertoire-based strategy s chooses the action associated with repertoire $\text{CreateRepertoire}(h)$.

We can use the P^{Rep} functions to define a formula that determines the EPRU for any repertoire-based strategy s . $\text{EPRU}_{\text{alt}}(s, R, r \mid G, \mathbf{S})$ is a recursive function for calculating the expected per-round utility of s :

$$\begin{aligned} \text{EPRU}_{\text{alt}}(s, R, r \mid G, \mathbf{S}) &= \sum_{a \in A} \sum_{v \in V} [P^{Rep}(\text{newaction}(R, v) \mid R, r, a, \mathbf{S}) \times \\ &\quad (\text{EV}_{\text{exp}}(r, U((a, (-, v)))) + \text{EPRU}_{\text{alt}}(s, \text{newaction}(R, v), r + 1 \mid G, \mathbf{S})) \\ &\quad + \sum_{m=1}^{|R|} P^{Rep}(\text{updaction}(R, v, m) \mid R, r, a, \mathbf{S}) \times \\ &\quad (\text{EV}_{\text{exp}}(r, U((a, (-, v)))) + \text{EPRU}_{\text{alt}}(s, \text{updaction}(R, v, m), r + 1 \mid G, \mathbf{S}))] \end{aligned}$$

where A is the set of possible actions and V is the set of possible action values.

However, like $\text{EPRU}_{\text{alt}}(s, h \mid G, \mathbf{S})$, $\text{EPRU}_{\text{alt}}(s, R, r \mid G, \mathbf{S})$ contains infinite recursion and is therefore not computable. We will deal with this problem as we did in Section 6.1, by introducing a depth-limited version. For ease of exposition we will introduce two "helper" functions

$$\begin{aligned} \text{EPRU}_{\text{alt}_{\text{new}}}^k(s, R, r, a, v \mid G, \mathbf{S}) &= P^{Rep}(\text{newaction}(R, v) \mid R, r, a, \mathbf{S}) \times \\ &\quad [\text{EV}_{\text{exp}}(r, U((a, (-, v)))) + \text{EPRU}_{\text{alt}}^{k-1}(s, \text{newaction}(R, v), r + 1 \mid G, \mathbf{S})] \end{aligned}$$

and

$$\begin{aligned} \text{EPRU}_{\text{alt}_{\text{upd}}}^k(s, R, r, a, v \mid G, \mathbf{S}) &= \sum_{m=1}^{|R|} P^{Rep}(\text{updaction}(R, v, m) \mid R, r, a, \mathbf{S}) \times \\ &\quad [\text{EV}_{\text{exp}}(r, U((a, (-, v)))) + \text{EPRU}_{\text{alt}}^{k-1}(s, \text{updaction}(R, v, m), r + 1 \mid G, \mathbf{S})] \end{aligned}$$

Now we can define

$$\text{EPRU}_{\text{alt}}^k(s, R, r \mid G, \mathbf{S}) = \begin{cases} 0, & \text{if } k = 0, \\ \sum_{a \in A} \sum_{v \in V} (\text{EPRU}_{\text{alt}_{\text{new}}}^k(s, R, r, a, v \mid G, \mathbf{S}) + \text{EPRU}_{\text{alt}_{\text{upd}}}^k(s, R, r, a, v \mid G, \mathbf{S})), & \text{otherwise.} \end{cases}$$

A proof that this formulation is equivalent to the version of $\text{EPRU}_{\text{alt}}^k$ from Section 6.1 follows.

Theorem 5 For all histories h_α , all repertoire-based strategies s , and all $k \geq 0$, if s' is a function from histories to actions where $s'(h) = s(\text{CreateRepertoire}(h))$ and $r = |h_\alpha|$, then $\text{EPRU}_{\text{alt}}^k(s, R, r | G, \mathbf{S}) = \text{EPRU}_{\text{alt}}^k(s', h | G, \mathbf{S})$.

Proof. We can prove this by using induction on k . For our base case, we will use $k = 0$, since $\text{EPRU}_{\text{alt}}^0(s, R, r | G, \mathbf{S}) = \text{EPRU}_{\text{alt}}^0(s', h | G, \mathbf{S}) = 0$ by definition.

For the inductive step, we will assume that Theorem 5 holds for some $k \geq 0$, and show that it also holds for $k + 1$. Recall from Section 6.1 that, in this case

$$\text{EPRU}_{\text{alt}}^{k+1}(s', h | G, \mathbf{S}) = \sum_{t \in T} P(h \circ t | h, s(h), \mathbf{S}) (\text{EV}_{\text{exp}}(r, U(t)) + \text{EPRU}_{\text{alt}}^k(s', h \circ t | G, \mathbf{S}))$$

Since T is simply the set of all action-percept pairs, we can instead write this as

$$\begin{aligned} \text{EPRU}_{\text{alt}}^{k+1}(s', h | G, \mathbf{S}) = \\ \sum_{a \in A} \sum_{v \in V} \left[\sum_{m=1}^{\mu} P(h \circ (a, (m, v)) | h, s(h), \mathbf{S}) \right. \\ \left. (\text{EV}_{\text{exp}}(r, U((a, (m, v)))) + \text{EPRU}_{\text{alt}}^k(s', h \circ (a, (m, v)) | G, \mathbf{S})) \right] \end{aligned}$$

where A is the set of possible actions and V the set of possible values. We also have¹⁴

$$\begin{aligned} \text{EPRU}_{\text{alt}}^{k+1}(s, R, r | G, \mathbf{S}) = \\ \sum_{a \in A} \sum_{v \in V} \left[P^{\text{Rep}}(\text{newaction}(R, v) | R, r, a, \mathbf{S}) \right. \\ (\text{EV}_{\text{exp}}(r, U((a, (-, v)))) + \text{EPRU}_{\text{alt}}^k(s, \text{newaction}(R, v), r + 1 | G, \mathbf{S})) \\ + \sum_{m=1}^{|R|} P^{\text{Rep}}(\text{updaction}(R, v, m) | R, r, a, \mathbf{S}) \\ \left. (\text{EV}_{\text{exp}}(r, U((a, (-, v)))) + \text{EPRU}_{\text{alt}}^k(s, \text{updaction}(R, v, m), r + 1 | G, \mathbf{S})) \right] \end{aligned}$$

Recall from Lemma 6 that P and P^{Rep} are *consistent* on h_α . We can combine this with our inductive hypothesis to show that the bracketed portions of the two equations above are equal.

Recall that when a repertoire encounters a new action, it does not store the action number m for that action. Thus, for any pair m and m' that are not already in h_α , we know that $\text{CreateRepertoire}(h \circ (a, (m, v))) = \text{CreateRepertoire}(h \circ (a, (m', v))) = \text{newaction}(R, v)$. Therefore, by our inductive hypothesis

$$\text{EPRU}_{\text{alt}}^k(s', h \circ (a, (m, v)) | G, \mathbf{S}) = \text{EPRU}_{\text{alt}}^k(s, \text{newaction}(R, v), r | G, \mathbf{S})$$

for all m not already in h_α . Notice that, by definition, there are $\mu - X(h)$ values of m that are not already in h_α . If we assume without loss of generality that the first action learned in h_α has label 1, the second has label 2, etc., then we can define β_{new} to be the quantity P and P^{Rep} are multiplied by when we learn something new:

$$\begin{aligned} \beta_{\text{new}} &= \text{EV}_{\text{exp}}(r, U((a, (-, v)))) + \text{EPRU}_{\text{alt}}^k(s', h \circ \langle a, X(h) + 1, v \rangle | G, \mathbf{S}) \\ &= \text{EV}_{\text{exp}}(r, U((a, (-, v)))) + \text{EPRU}_{\text{alt}}^k(s', h \circ \langle a, X(h) + 2, v \rangle | G, \mathbf{S}) \\ &\dots \\ &= \text{EV}_{\text{exp}}(r, U((a, (-, v)))) + \text{EPRU}_{\text{alt}}^k(s', h \circ (a, (\mu, v)) | G, \mathbf{S}) \\ &= \text{EV}_{\text{exp}}(r, U((a, (-, v)))) + \text{EPRU}_{\text{alt}}^k(s, \text{newaction}(R, v), r | G, \mathbf{S}) \end{aligned}$$

¹⁴Recall that function U simply calculates the utility of performing the given action, and does not depend on the action number. Thus $U(a, m, v) = U(a, -, v)$ for any legal m .

Similarly, the inductive hypothesis also tells us that

$$\text{EPRU}_{\text{alt}}^k(s', h \circ (a, (m, v)) | G, \mathbf{S}) = \text{EPRU}_{\text{alt}}^k(s, \text{updaction}(R, v, m), r | G, \mathbf{S})$$

for all m that are already in h_α . Thus, we can also define β_m to be the quantity P and P^{Rep} are multiplied by when we update our information on action m :

$$\begin{aligned} \beta_m &= \text{EV}_{\text{exp}}(r, U((a, (-, v)))) + \text{EPRU}_{\text{alt}}^k(s', h \circ (a, (m, v)) | G, \mathbf{S}) \\ &= \text{EV}_{\text{exp}}(r, U((a, (-, v)))) + \text{EPRU}_{\text{alt}}^k(s, \text{updaction}(R, v, m), r | G, \mathbf{S}) \end{aligned}$$

for $m = 1, \dots, X(h)$.

We can now rewrite $\text{EPRU}_{\text{alt}}^{k+1}(s', h | G, \mathbf{S})$ and $\text{EPRU}_{\text{alt}}^{k+1}(s, R, r | G, \mathbf{S})$ as

$$\begin{aligned} \text{EPRU}_{\text{alt}}^{k+1}(s', h | G, \mathbf{S}) &= \sum_{a \in A} \sum_{v \in V} \left[\sum_{m=X(h)+1}^{\mu} P(h \circ (a, (m, v)) | h, s(h), \mathbf{S}) \beta_{\text{new}} \right. \\ &\quad \left. + \sum_{m=1}^{X(h)} P(h \circ (a, (m, v)) | h, s(h), \mathbf{S}) \beta_m \right] \end{aligned}$$

and

$$\begin{aligned} \text{EPRU}_{\text{alt}}^{k+1}(s, R, r | G, \mathbf{S}) &= \sum_{a \in A} \sum_{v \in V} [P^{Rep}(\text{newaction}(R, v) | R, r, a, \mathbf{S}) \beta_{\text{new}} \\ &\quad + \sum_{m=1}^{|R|} P^{Rep}(\text{updaction}(R, v, m) | R, r, a, \mathbf{S}) \beta_m] \end{aligned}$$

Equations 21 and 22 tell us that regardless of the values of a and v ,

$$\sum_{m=X(h)+1}^{\mu} P(h \circ (a, (m, v)) | h, s(h), \mathbf{S}) = P^{Rep}(\text{newaction}(R, v) | R, r, a, \mathbf{S})$$

and

$$\sum_{m=1}^{X(h)} P(h \circ (a, (m, v)) | h, s(h), \mathbf{S}) = \sum_{m=1}^{|R|} P^{Rep}(\text{updaction}(R, v, m) | R, r, a, \mathbf{S})$$

Therefore, $\text{EPRU}_{\text{alt}}^{k+1}(s', h | G, \mathbf{S}) = \text{EPRU}_{\text{alt}}^{k+1}(s, R, r | G, \mathbf{S})$. This completes the induction. \square

B.0.5 Repertoire-Based Algorithm

Now that we have a formula for computing the EPRU of a repertoire-based strategy, and we know that using repertoires rather than histories to calculate EPRU gives us the same results, we can update our algorithm to use repertoires. The new algorithm will be almost identical to Algorithm 1, except that using repertoires rather than histories will allow us to reduce our number of recursive calls by half. Let R'_{Obs} be the set of repertoires for which $P^{Rep}(R'_{\text{Obs}} | R, r, \text{Obs}) > 0$, and define R'_{Inv} and R'_{X_i} similarly. Note that, if R contains m different actions,

$$R'_{\text{Obs}} \subseteq R'_{\text{Inv}} \cup \bigcup_{i=1}^m R'_{X_i} \tag{23}$$

Algorithm 4 Produce strategy s that maximizes $\text{EPRU}_{\text{alt}}^k(s, R \mid G, \mathbf{S})$, given initial repertoire R , and set of possible utility values V .

$\text{Strat}(R, r, k, V, \mathbf{S})$

```

1: if  $k = 0$  then
2:   return 0
3: end if
4: Let  $U_{\max} = 0$ 
5: Let  $s_{\max} = \text{null}$ 
6: Let  $U_{\text{Obs}} = 0$ 
7: for each action  $a \in \{X_1, \dots, X_\mu, \text{InV}\}$  do
8:   Let  $U_{\text{temp}} = 0$ 
9:   Let  $s_{\text{temp}} = \text{null}$ 
10:  for each value  $v \in V$  do
11:    Let  $t = \langle v, 1 \rangle$ 
12:    if  $\exists i : (a = X_i)$  then Let  $R' = \text{age}(R \setminus \{\langle v_i, \gamma_i \rangle\}) \cup t$  and  $p = P^{\text{Rep}}(R' \mid R, r, X_i, \mathbf{S})$ 
13:    else Let  $R' = \text{age}(R) \cup t$  and  $p = P^{\text{Rep}}(R' \mid R, r, \text{InV}, \mathbf{S})$ 
14:    Let  $p_{\text{Obs}} = P^{\text{Rep}}(R' \mid R, r, \text{Obs}, \mathbf{S})$ 
15:    if  $p + p_{\text{Obs}} > 0$  then
16:      Let  $\{S', U'\} = \text{Strat}(R', r + 1, k - 1, V, \mathbf{S})$ 
17:      if  $p_{\text{Obs}} > 0$  then  $U_{\text{Obs}} = U_{\text{Obs}} + p_{\text{Obs}} \cdot U'$ 
18:      if  $p > 0$  then
19:         $s_{\text{temp}} = s_{\text{temp}} \cup S'$ 
20:        if  $\exists i : (a = X_i)$  then  $U_{\text{temp}} = U_{\text{temp}} + p \cdot (\text{EV}_{\text{exp}}(r, v) + U')$ 
21:        else  $U_{\text{temp}} = U_{\text{temp}} + p \cdot U'$ 
22:      end if
23:    end if
24:  end for
25:  if  $U_{\text{temp}} > U_{\max}$  then
26:     $U_{\max} = U_{\text{temp}}$ 
27:     $s_{\max} = s_{\text{temp}} \cup \langle R, a \rangle$ 
28:  end if
29: end for
30: if  $U_{\text{Obs}} > U_{\max}$  then
31:   $U_{\max} = U_{\text{Obs}}$ 
32:   $s_{\max} = s_{\text{temp}} \cup \langle R, \text{Obs} \rangle$ 
33: end if
34: return  $\{s_{\max}, U_{\max}\}$ 

```

In other words, since repertoires do not need to remember what actions our agent performed, choosing X_1 produces the same repertoire as choosing **Obs** and observing action 1. Similarly, choosing **InV** and **Obs** can also produce the same repertoires, if both actions happen to tell us about the same action. However, there is no action we could encounter through observing that we could not encounter through either innovating or exploiting. Therefore, if we save the results of the recursive calls to calculate the utility of **InV** and X_1, \dots, X_m , we can compute the utility of **Obs** without any additional recursion. This cuts the branching factor of our algorithm in half, from $(2m + 2)v$ to $(m + 1)v$, which reduces the size of the search tree by a factor of 2^k for search depth k , without any impact on accuracy. Algorithm 4 is the complete algorithm.

C Number of Cultaptation Strategies

In Cultaptation as defined on the tournament web site, each game includes 10,000 rounds, 100 agents, 100 exploitation actions, and the actions `Inv` and `Obs`. Let S be the set of all pure Cultaptation strategies, and S' be the set of all strategies such that the first 100 moves are `Inv`, and all subsequent moves are exploitation actions. Then any lower bound on S' is a loose lower bound on S .

Suppose an agent uses a strategy in S' . If it survives for the first 100 rounds of the game, it will learn values for all 100 of the exploitation actions. There are $100!$ different orders in which these actions may be learned, and for each action there are 100 possible values; hence there are 100^{100} possible combinations of values. Thus after 100 `Inv` moves, the number of possible histories is $100! \times 100^{100}$. All subsequent moves by the agent will be exploitations; and it is possible (though quite unlikely!) that the agent may live for the remaining 9,900 rounds of the game. Thus each of the above histories is the root of a game tree of height $2 \times 9,900$. In this game tree, each node of even depth is a choice node (each branch emanating from the node corresponds to one of the 100 possible exploitation actions), and each node of odd depth is a value node (each branch emanating from the node corresponds to one of the 100 different values that the chosen action may return). Since there are $100! \times 100^{100}$ of these game trees, the total number of choice nodes is

$$100! * 100^{100} \sum_{d=0}^{9899} (100^2)^d > 9.3 \times 10^{39953}.$$

If we use the conventional game-theoretic definition that a pure strategy s must include a choice of action at each choice node, regardless of whether the choice node is reachable given s , then it follows that

$$|S'| > 100^{9.3 \times 10^{39953}}.$$

If we use the definition used by game-tree-search researchers, in which a pure strategy only includes a choice of action at each choice node that is *reachable* given s , then the number of reachable choice nodes given s is

$$100! * 100^{100} \sum_{d=0}^{9899} 100^d > 9.4 \times 10^{20155},$$

so

$$|S'| > 100^{9.4 \times 10^{20155}}.$$