ABSTRACT

Title of dissertation: GENERATION AND ANALYSIS OF STRATEGIES IN AN EVOLUTIONARY SOCIAL LEARNING GAME

James Ryan Carr, Doctor of Philosophy, 2013

Dissertation directed by: Professor Dana Nau
Department of Computer Science

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 dissertation I present an extensive set of mathematical and simulation results for Cultaptation, which is one of the best-known such games.

I derive a formula for measuring a strategy's expected reproductive success, and provide algorithms to compute near-best-response strategies and near-Nash equilibria. Some of these algorithms are too complex to run quickly on larger versions of Cultaptation, so I also show how they can be approximated to be able to handle larger games, while still exhibiting better performance than the current best-known Cultaptation strategy for such games. 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 prefer social learning most of the time, but will have ways of quickly detecting structural shocks, so that they can switch quickly to individual learning 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 individual learning; and agrees with recent experiments by others on human subjects that also challenge the conventional wisdom.

GENERATION AND ANALYSIS OF STRATEGIES IN AN EVOLUTIONARY SOCIAL LEARNING GAME

by

James Ryan Carr

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2012

Advisory Committee:
Professor Dana Nau, Chair/Advisor
Professor Michele Gelfand
Professor Mohammad Hajiaghayi
Professor Donald Perlis
Professor James Reggia

Table of Contents

Chapter 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 [**?**], and it also is important for many other animal species [**?**, **?**, **?**]. Such learning usually involves evaluating the outcomes of others' actions, rather than indiscriminate copying of others' behavior [**?**, **?**], 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 [**?**, **?**, **?**, **?**, **?**].

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

- Each agent has three kinds of possible actions: innovation, observation, and exploitation. These are highly simplified analogs of the following real-world

[1]NOTE: We are not affiliated with the tournament or with the Cultaptation project.

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.

The organizers of the tournament have conducted a thorough analysis of its results that made many observations [**?**]; perhaps the most interesting result was that the top-performing strategies in the tournament all relied on social learning almost exclusively, performing individual learning only as a bootstrapping mechanism in the

[2] For 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*.

first rounds of the game. This was considered surprising because prior theoretical studies of social learning suggest that individual learning should play a prominent role.

A second Cultaptation tournament began in February 2012, and the entries are still being evaluated at the time of this writing.[3] Unfortunately, one of the inherent disadvantages of a tournament format is that its results provide evidence that the winning contestants are superior to the other contestants; this does prove whether the winning contestants use the best possible methods.

In this work, we attempt to address this problem by developing analytical methods capable of analyzing evolutionary environments with social learning, and using these methods to study Cultaptation to see what types of strategies are effective. We present the following results in this work:

First, we provide a strategy-generation algorithm that can construct a strategy that is within any given error bound $\epsilon > 0$ of a best response to a given set of competing strategies (Section **??**).

We then present the Cultaptation Strategy Learning Algorithm (CSLA), which uses the strategy-generation algorithm in an iterative self-improvement loop to attempt to find a strategy that is a near-best response to itself, and is therefore a symmetric near-Nash equilibrium (Section **??**). Finding such a strategy is desirable because it should be able to perform well against any set of competing strategies

[3]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 work does not deal with these additions, although we may address them in future work.

(i.e., any strategies it plays against in a tournament setting).

Finally, we provide experimental results outlining the generation of an approximate Nash equilibrium strategy, s_{self}, in a small version of Cultaptation, and a performance comparison (in the smaller environment) between s_{self} and EVChooser, a known good strategy from the Cultaptation tournament (Section **??**).[4] These experiments show that s_{self} is able to outperform EVChooser, and provide several insights into the characteristics of good Cultaptation strategies. For example, the experiments show that s_{self} observes and exploits most of the time, but switches quickly to innovation when a structural shock occurs, switching back to observation and exploitation once it has learned how to respond to the shock. This conflicts with the conventional wisdom [**?**, **?**] 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 [**?**] and some recent experimental results on human subjects [**?**].

These results provide strong support for the following hypotheses about the best strategies for Cultaptation and similar games: First, 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. Second, it is likely that the best strategies will observe and exploit most of the time, but will

[4]Further experiments, included in the supplemental material, demonstrate that these results extend to the full-size game of Cultaptation as well.

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.

In addition to these insights about Cultaptation and social learning in general, our work also shows how the analysis and algorithms outlined above could be adapted to apply to future evolutionary games that, like Cultaptation, are significantly more complex than classical evolutionary games. As researchers in the field of evolutionary game theory continue to study more complex phenomena like social learning, it is likely that these weaker assumptions will be needed more frequently, and so techniques like the ones presented here will be necessary more often.

Chapter 2

Background

2.1 Cultaptation Social-Learning Game

This section gives a more detailed description of the Cultaptation social learning game, adapted from the official description [**?**]. 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 ($\mathsf{X}_1, \ldots, \mathsf{X}_M$, where M 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. 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 proposal, but it is straightforward to extend my algorithms to take this difference into account.

Example 1. Consider two strategies: the innovate-once strategy (here-

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 2.1: Action sequences from Example **??**, and their utilities.

after 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 **??**; 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 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 I will show shortly).

Formally, everything that an agent α knows about each round can be described

by an *action-percept pair*, $(a, (m, v))$, where $a \in \{\mathsf{Inv}, \mathsf{Obs}, \mathsf{X}_1, \ldots, \mathsf{X}_M\}$ is the action that α chose to perform, and (m, v) is the percept returned by the action. More specifically, $m \in \{\mathsf{X}_1, \ldots, \mathsf{X}_M, \emptyset\}$ 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, $h_\alpha = \langle (a_1, (m_1, v_1)), \ldots, (a_k, (m_k, v_k)) \rangle$. As a special case, the empty (initial) history is $\langle\rangle$.

Example 2. The history for $\mathsf{I2O}$ in Example **??** is:

$$h_{\mathsf{I2O}} = \langle (\mathsf{Inv}, (\mathsf{X}_3, 8)), (\mathsf{Inv}, (\mathsf{X}_4, 5)), (\mathsf{Obs}, (\mathsf{X}_1, 3)), (\mathsf{X}_3, (\mathsf{X}_3, 8)), \ldots \rangle$$

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 $h_\alpha = \langle p_1, p_2, \ldots, p_k \rangle$, where each p_i is some action-percept pair, I let $h_\alpha[i] = p_i$, and $h_\alpha[i, \ldots, j]$ be the subhistory $\langle p_i, \ldots, p_j \rangle$.

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;[1] but any particular Cultaptation game is a competition among a much smaller set of strategies **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,

$$\mathbf{S} = \{\text{the strategies chosen by the contestants}\}. \tag{2.1}$$

Each strategy in **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 [**?**].

The replication dynamics work as follows. On each round, each agent has a 2% chance of dying. Therefore, we also include a parameter d in my 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

[1] The number of possible mixed strategies is, of course, infinite. But even if we consider only pure strategies, the number is quite huge. For a 10,000-round Cultaptation game of the type used in the Cultaptation tournament, a loose lower bound on the number of pure strategies is $100^{9.4\times10^{20155}}$ [**?**]. In contrast, it has been estimated [**?**] that the total number of atoms in the observable universe is only about 10^{78} to 10^{82}.

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.

> **Example 3.** Again looking at the sequences of actions in Table **??**, 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 [**?**], 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 $s_1, \ldots, s_k$ 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. Carr et al. [**?**] have demonstrated that any strategy maximizing an agent's expected per-round utility (explained in Section **??**) will also maximize its reproduction. Therefore, in this work we will focus on finding strategies with maximal expected per-round utility.

2.2 Motivating Discussion

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

- Sections **??** and **??** 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

proposal, 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 game-tree search algorithms. Section **??** explains some issues that are important for finding best-response strategies in Cultaptation.

2.2.1 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, 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.[2]

> **Example 4. Structural shocks:** Figure **??** 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 $\emptyset$,[3] in order to obtain an exploitation action X_i which the agent will use in all subsequent rounds.

[2] We have borrowed this term from the Economics literature, where it has an analogous meaning (e.g., [**?**, **?**]).

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

Round	X_1	X_2	X_3	X_4	A1	A2	A3
1	2	4	1	9	N/A	N/A	$(\mathsf{Obs}, (\emptyset, \cdot))$
2	2	4	1	9	N/A	N/A	$(\mathsf{Inv}, (\mathsf{X}_4, 9))$
3	2	4	1	9	Birth	N/A	$(\mathsf{X}_4, (\cdot, 9))$
4	5	4	1	9	$(\mathsf{Obs}, (\mathsf{X}_4, 9))$	Birth	$(\mathsf{X}_4, (\cdot, 9))$
5	5	2	1	9	$(\mathsf{X}_4, (\cdot, 9))$	$(\mathsf{Obs}, (\mathsf{X}_4, 9))$	$(\mathsf{X}_4, (\cdot, 9))$
6	1	2	1	9	$(\mathsf{X}_4, (\cdot, 9))$	$(\mathsf{X}_4, (\cdot, 9))$	$(\mathsf{X}_4, (\cdot, 9))$
7	1	2	8	9	$(\mathsf{X}_4, (\cdot, 9))$	$(\mathsf{X}_4, (\cdot, 9))$	Death
8	1	2	8	1	$(\mathsf{X}_4, (\cdot, 1))$	$(\mathsf{X}_4, (\cdot, 1))$	N/A
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Figure 2.1: 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.

> 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 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.

2.2.2 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. [**?**] 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.

2.2.3 Best-Response Strategies in Cultaptation

A widely used technique for getting insight about a game (e.g., see [**?**]) 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. This is because the strategy profile will change many times during the course of the game, as agents die and other agents are born to take their places.

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 **??** 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 $\pi_{\mathsf{Obs}}(a|h_\alpha, \mathbf{S})$ 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 the agent might take, which will allow us to determine the best response to **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 work.

Chapter 3

Related Work

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

3.1 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. [**?**]. 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 [**?**, **?**]. However, this result is consistent with observations made during my own experiments, in which the ϵ-best-response strategy rarely did innovation (see Section **??**).

In previous work, Carr et al. showed how to compute optimal strategies for a highly simplified versions of the Cultaptation social learning game [**?**]. 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 [**?**]. 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 my 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 [**?**]. In

this case, analyzing the strategies found by my algorithm allowed me to draw the same conclusions about what works well. This gives more evidence that the conventional wisdom on social learning [**?**, **?**] 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 [**?**]. 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 [**?**]. Game theoretic approaches have also been used to explore this subject, but it is still ongoing research [**?**, **?**]. 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 [**?**].

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 [**?**]. 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.

3.2 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 [**?**]. 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. I also show in Section **??** 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 [**?**, **?**]. Multi-armed bandit problems have previously been used to study the tradeoff between exploitation and exploration in learning environments [**?**, **?**].

As discussed later in Section **??**, finding a best-response strategy in Cultaptation is basically equivalent to finding an optimal policy for a Markov Decision Process. Consequently, my 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 [**?**]. Both algorithms use the discount factor of the MDP (which, in the case of Cultaptation, 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 mine

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. Mine, 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 the 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 Cultaptation, which have exponentially large state spaces. Several approaches for *near-optimal* control, which produces a policy within some ϵ of optimal, have been developed [**?**, **?**, **?**].

Chapter 5

Strategy Generation Algorithms

This chapter describes the algorithms used to generate near-best response strategies for Cultaptation, and presents experimental studies in which near-best response strategies are tested against a known good strategy from the Cultaptation tournament.

5.1 Finding an ϵ-Best Response Strategy

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

5.1.1 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 **??**) 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. $\mathbf{S}_{-\alpha} = \{s_1, ..., s_{\alpha-1}, s_{\alpha+1}, ..., s_\ell\}$. Strategy s_{opt} is a *best response* to $\mathbf{S}_{-\alpha}$ if and only if for any other strategy s',

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

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 $\text{EPRU}(s \mid G, \mathbf{S}_{-\alpha} \cup s)$ is arbitrarily close to $\text{EPRU}(s_{\text{opt}} \mid G, \mathbf{S}_{-\alpha} \cup s_{\text{opt}})$. 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 $\text{EPRU}(s_\alpha \mid G, \mathbf{S}_{-\alpha} \cup s_\alpha)$ is within ϵ of $\text{EPRU}(s_{\text{opt}} \mid G, \mathbf{S}_{-\alpha} \cup s_{\text{opt}})$.

5.1.2 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. I will use this fact in deriving a bound on $\text{EPRU}_{\text{alt}}(s, h_\alpha \mid G, \mathbf{S})$ for a given strategy and a history h_α of length l.

Recall from Equations **??** and **??** that:

$$\text{EV}_{exp}(r, v) = v \sum_{i=r}^{\infty} \frac{1}{i}(1-d)^{i-1} = v \left(\frac{\ln d}{d-1} - \sum_{i=1}^{r-1} \frac{1}{i}(1-d)^{i-1} \right) \tag{5.1}$$

where $\text{EV}_{exp}(r, v)$ 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 EPRU($s_\alpha \mid G, \mathbf{S}$) 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 EPRU($s_\alpha \mid G, \mathbf{S}$), 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:

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

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

$$G(l,v) = v\left(\frac{(1-d)^l}{d} - \underbrace{l\left(\frac{\ln d}{d-1} - \sum_{n=1}^{l}\frac{1}{n}(1-d)^{n-1}\right)}_{a}\right) \tag{5.2}$$

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 **??**), 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). I use this fact to bound the depth limited expected per-round utility computation.

Theorem 1. *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,*

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

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. □

Theorem **??** states that $G(l, v_{max})$ is the highest possible contribution to the total expected per-round utility (i.e., $\mathrm{EPRU}_{\mathrm{alt}}(s_\alpha, \langle\rangle \mid G, \mathbf{S})$) 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.

5.1.3 Determining How Far to Search

In this section I 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 1. $G(l, v) \leq v(1-d)^l/d$.

Proof. The lemma follows from noting that part (a) of Equation **??** 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 $G(l, v) = v(\frac{(1-d)^l}{d} - w) \leq v\frac{(1-d)^l}{d}$, since w is always non-negative. □

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

$$k = log_{(1-d)}\left(\frac{d\epsilon}{v_{max}}\right), \tag{5.3}$$

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 2. *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* $\mathrm{EPRU}^k_{\mathrm{alt}}(s_\alpha, \emptyset \mid G, \mathbf{S}_{-\alpha} \cup s_\alpha)$*, then s_α is an* ϵ-best response *to* $\mathbf{S}_{-\alpha}$.

Proof. Let s_{opt} be the strategy with the maximal value of $\mathrm{EPRU}(s_{\mathrm{opt}} \mid G, \mathbf{S}_{-\alpha} \cup s_{\mathrm{opt}})$. By Theorem **??**, 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 $|\,\mathrm{EPRU}(s_{\mathrm{opt}} \mid G, \mathbf{S}_{-\alpha} \cup s_{\mathrm{opt}}) - \mathrm{EPRU}(s_\alpha \mid G, \mathbf{S}_{-\alpha} \cup s_\alpha)| \leq \epsilon$. Therefore, s_α is an ϵ-best response. □

5.1.4 Algorithm

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

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

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

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

if $k = 0$ **then**
  return 0
end if
Let $U_{\max} = 0$
Let $s_{\max} = null$
for each action $a \in \{\mathsf{Inv}, \mathsf{Obs}, \mathsf{X}_1, \ldots, \mathsf{X}_M\}$ **do**
  Let $U_{\text{temp}} = 0$
  Let $s_{\text{temp}} = \langle h_\alpha, a \rangle$
  for each action $m \in \{1, \ldots, M\}$ **do**
    for each value $v \in V$ **do**
      Let $t = (a, (m, v))$
      Let $p = P(h_\alpha \circ t | h_\alpha, a, \mathbf{S}_{-\alpha})$
      if $p > 0$ **then**
        Let $\{S', U'\} = \text{Strat}(h_\alpha \circ t, k-1, V, \mathbf{S}_{-\alpha})$
        $s_{\text{temp}} = s_{\text{temp}} \cup S'$
        $U_{\text{temp}} = U_{\text{temp}} + p\,(\text{EV}_{exp}(|h_\alpha \circ t|, U(t)) + U')$
      end if
    end for
  end for
  if $U_{\text{temp}} > U_{\max}$ **then**
    $U_{\max} = U_{\text{temp}}$
    $s_{\max} = s_{\text{temp}}$
  end if
end for
return $\{s_{\max}, U_{\max}\}$

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. Figure **??** provides an example of the kind of tree searched by this algorithm. For each possible action $a \in \{\mathsf{Inv}, \mathsf{Obs}, \mathsf{X}_1, \ldots, \mathsf{X}_M\}$ 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 $\text{EPRU}^k_{\text{alt}}(s, h_\alpha \mid G, \mathbf{S}_{-\alpha} \cup s)$ for all possible strategies s, returning the strategy maximizing $\text{EPRU}^k_{\text{alt}}$ as well as the maximal value of $\text{EPRU}^k_{\text{alt}}$.

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

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

A proof of this proposition is presented in [**?**].

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 3. *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 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 **??** and Proposition **??**. □

We also have the following.

Corollary 1. *Given available strategies other than our own* $\mathbf{S}_{-\alpha}$ *and a set of values* V*, let* s_k *be the strategy returned by Strat*$(\emptyset, k, V, \mathbf{S}_{-\alpha})$*. Then* $\lim_{k\to\infty} s_k$ *is a best*

response to $\mathbf{S}_{-\alpha}$.

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

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

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

5.1.5 Implementation

In this section I discuss modifications that improve the running time of Algorithm **??** without any loss in accuracy. Section **??** discusses techniques for state aggregation that cut the branching factor of the algorithm in half. Section **??** discusses the representation of π_{Obs}, and Section **??** discusses caching and pruning.

State Aggregation

If the pseudocode for Algorithm **??** 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 $h_\alpha = \langle(\mathsf{Inv}, (1, 10))\rangle$ and $h'_\alpha = \langle(\mathsf{Inv}, (2, 10))\rangle$ 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

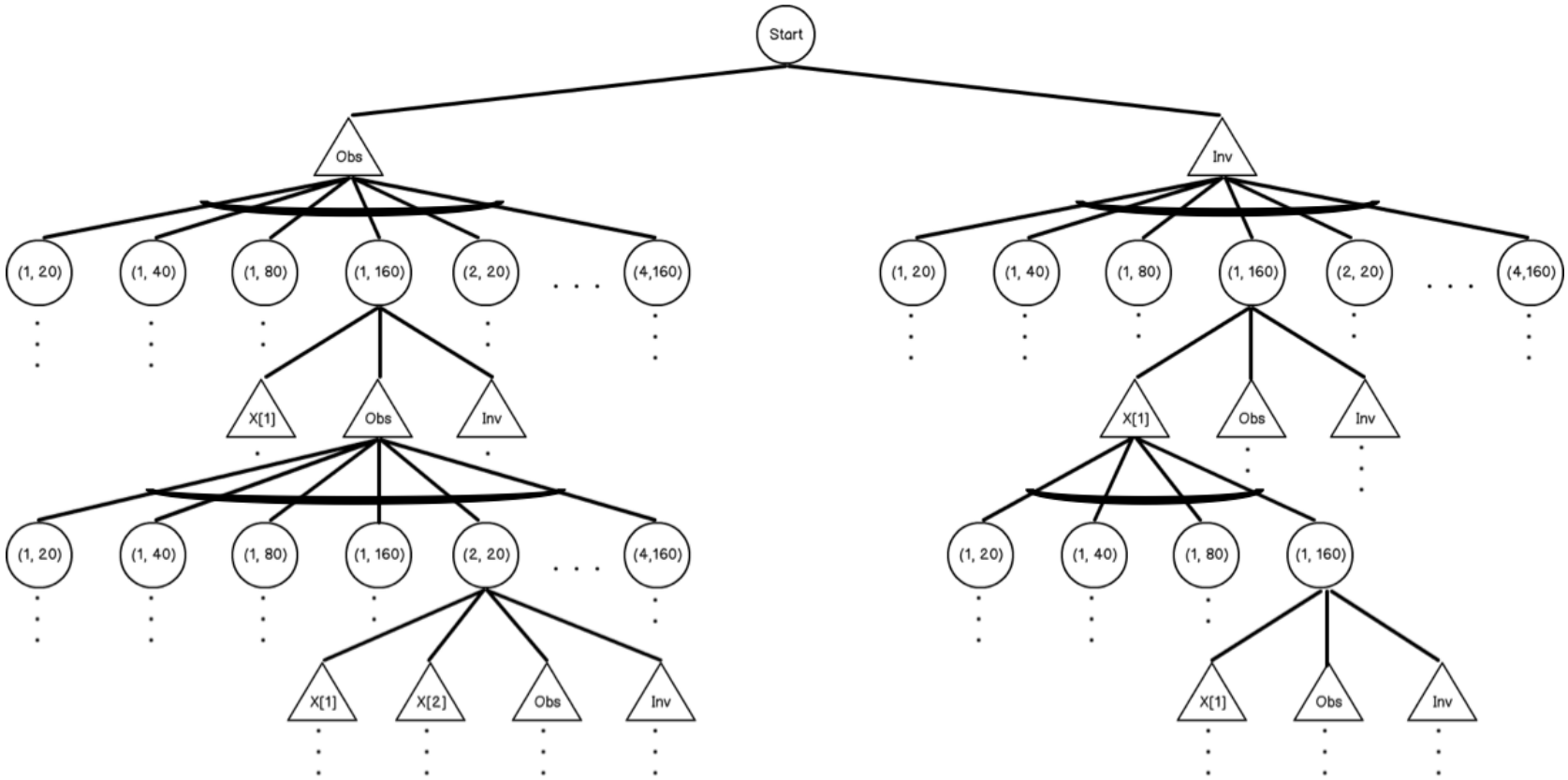


Figure 5.1: An illustration of a portion of the search performed by Algorithm **??** on a very small version of Cultaptation, with four actions and a set of possible action values equal to $\{20, 40, 80, 160\}$. Circle nodes indicate possible observations made by the agent, and triangle nodes indicate possible choices made by the agent (which have a random outcome, indicated by the line through the branches leading to its children). Ellipses indicate branches or subtrees omitted from the figure. The left side of the illustration shows how more actions can become available to the agent as the game progresses, while the right side shows how potentially-changing values of actions must be modeled in the tree. Several techniques outlined in Section **??** can reduce the size of the tree that needs to be searched in the implementation of this algorithm.

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 **??** 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 the 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 **??** considers a history h_α, it makes one recursive call for each possible action-percept pair $(a, (m, v))$ that can be executed at h_α. There are $2M$ such pairs for each history; if the agent knows how to exploit j actions, then it can innovate any of the $M - j$ actions it does not know, and it can observe any of the M 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 $2Mv$. Since the algorithm recurses to depth k, the running time for Algorithm **??** is $O((2Mv)^k)$. 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 Mv, and the total running time is $O((Mv)^k)$, which improves upon the original running time by a factor of 2^k.

Representing π_{Obs}

For the formal proofs, I treated π_{Obs} as a black box that, when given the 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. Algorithm **??** (introduced in Section **??**) would also need to run a prohibitively large number of simulations to get enough samples to generate a new π_{Obs} of this type.

Therefore, as an approximation, my 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 the experimental results (particularly those dealing with iterative computation in Section **??**) that this form of π_{Obs} is still able to produce good strategies.

Caching and Pruning

Since the 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 [**?**].

I also use another well-known method for avoiding unnecessary evaluation of states, namely branch-and-bound pruning [**?**, **?**], which can be summarised as follows. Before we compute the expected per-round utility of a given action, we check 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).

There are no theoretical guarantees on runtime reduction using these techniques, but we will see in Section **??** that the combination of pruning and caching allows the algorithm to avoid evaluating significant portions of the state space in the environments I tested.

5.2 Cultaptation Strategy Learning Algorithm

Until now I have assumed that Algorithm **??** 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}, 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, I 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 $\pi_{\mathsf{Obs}} = \pi_{\mathsf{Inv}}$. 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, $\text{stratDiff}(s, s_{\text{old}}) =$

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

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

1: Let $\pi_{\mathsf{Obs}} = \pi_{\mathsf{Inv}}$.
2: $s = \emptyset$.
3: **repeat**
4: Let $s_{\text{old}} = s$.
5: Let $V = \{\pi_{\mathsf{Inv}}, \pi_{\mathsf{Obs}}\}$.
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. $\pi_{\mathsf{Obs}}(v)$ = fraction of the time v was exploited in the records).
9: **until** $\text{stratDiff}(s, s_{\text{old}}) < \tau$
10: **return** s.

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 **??**.

Properties of the strategy. CSLA as presented here is a "best-effort" algorithm in the following sense: If CSLA converges to a strategy s, we know that it is an approximation of a symmetric Nash equilibrium strategy, but we do not know (i) whether or not CSLA will converge for a given environment, or (ii) how close to the true Nash equilibrium s is. The improved version of CSLA proposed in Chapter 5 will address these issues.

In my experimental studies (see Section **??**), 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 an 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.[1]

5.2.1 Implementation Details

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

Representing π_{Obs}

My implementation of CSLA uses the same representation of π_{Obs} as my implementation of Algorithm **??** does. In other words, it assumes π_{Obs} has the same

[1] Among 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 I hope to do it in my future research.

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.

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 **??**, strategy s is trained by playing against itself in a series of simulated games in step 7. However, in the implementation of CSLA the agents involved in the games in step 7 are 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. I test this hypothesis experimentally, in Section **??**.

5.3 Experimental Results

In this section I present my experimental results.

Section **??** presents a series of experiments comparing two strategies generated with my Cultaptation Strategy Learning Algorithm to a known good strategy used in the international Cultaptation tournament. I 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 **??** and **??**). Finally, I perform an in-depth qualitative analysis of all three strategies and highlight the differences in behavior that give my learned strategies an advantage (Section **??**).

5.3.1 Experiments with the Cultaptation Strategy Learning Algorithm

The objective of my second experiment was to examine the performance of strategies produced by the Cultaptation Strategy Learning Algorithm (Algorithm **??** in Section **??**), and the importance of the environment (see Section **??**) used to train these strategies. Specifically, I was 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 7 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 7) can do at repelling the invader.

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

For the known good strategy I 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 [**?**]. We chose EVChooser because (1) it has been shown to be a competitive strategy, (2) its source code was readily available to me (unlike the other successful strategies from the Cultaptation tournament), and (3) it could be tuned to perform well in the Cultaptation environments I 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 my experiments, I believe EVChooser is representative of most of the high-performing strategies from the tournament. Nearly all of the strategies described in the tournament report [**?**] 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 type 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.[2] Unlike my CSLA algorithm, none of the strategies in the tournament conducted lookahead search.

For this experiment, I 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 my strategy generating algorithm, this is the largest environment (i.e., smallest probability of death, highest number of actions and action values) for which the algorithm could compute full strategies in a reasonable amount of time.

Convergence and Consistency of CSLA

As part of this work, I have developed a Java implementation of Algorithm **??** that allows one to specify the type of game to be used for the simulation in Step 7, and 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.

[2] *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 my version of the game

While there are no theoretical guarantees that the strategies produced by Algorithm **??** will converge, the algorithm's similarity to policy iteration [**?**] led me to suspect that the they would converge. Also, since CSLA is greedy, i.e., it selects the best response strategy at each step of the iteration, I was interested in seeing whether the strategy it found represented a local maximum or a global one.

I designed a simple experiment to see how these issues would play out when generating s_{self} and s_{EVC}: I 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 **??**, and I used this modified program to generate 100 alternate versions of s_{self} and s_{EVC}. I then compared these alternates to the original s_{self} and s_{EVC} using stratDiff. In the case of s_{self}, I found that all 100 alternate versions were identical to the original. In the case of s_{EVC}, I 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 CSLA converge for the environment I am 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 estimate how different s_{self} and s_{EVC} are, I ran $\text{stratDiff}(s_{\text{self}}, s_{\text{EVC}})$ and found it to be 0.27. This means that training a strategy against an external, fixed strategy in Algorithm **??** 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,

Table 5.1: 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%

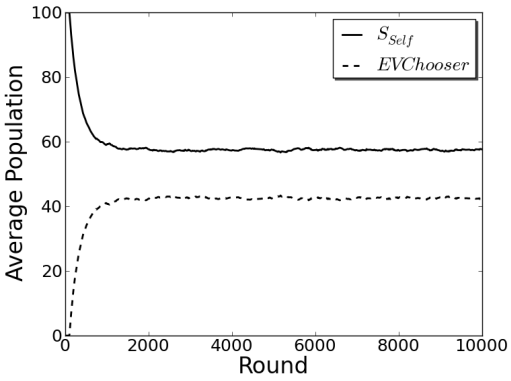
see Section **??**.

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

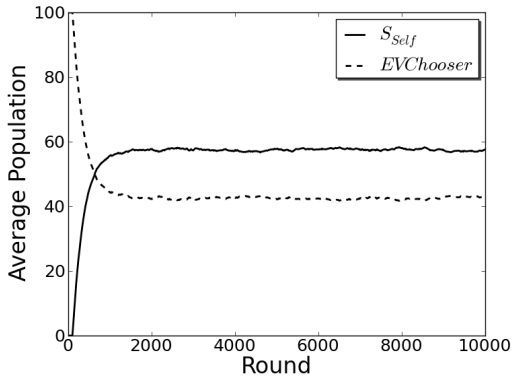
I played both of the generated strategies, s_{self} and s_{EVC}, against EVChooser for 20,000 games – in 10,000 games, the generated strategy was defending against an invading population of EVChooser agents, and in 10,000 games the roles were reversed, with the generated strategy invading and EVChooser defending. I recorded the population of each strategy on every round, as well as the winner of every game.[3] The populations in an individual game were extremely noisy, as seen in Figure **??**(e), however by averaging the populations over 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 **??**(a–d), while the win rates for each match-up are presented in Table **??**.

In Figure **??** we see that, on average, the strategies generated by Algorithm **??** 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

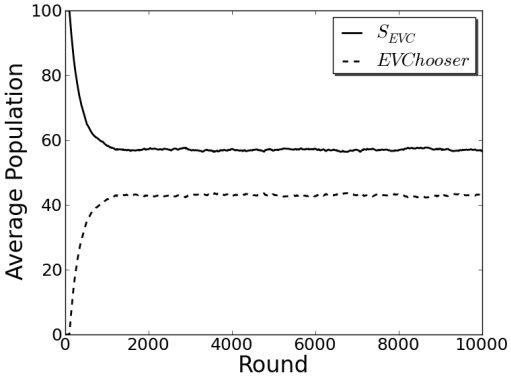
[3]Recall that the winner of a Cultaptation game is the strategy with the highest average population over the last quarter of the game.



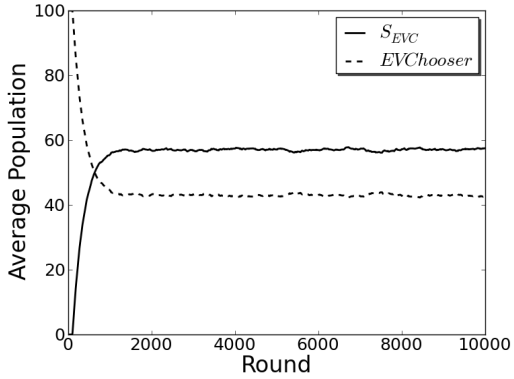
a) EVChooser invading s_{self}



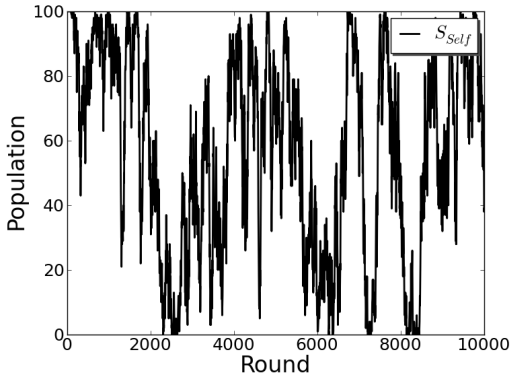
b) s_{self} invading EVChooser



c) EVChooser invading s_{EVC}



d) s_{EVC} invading EVChooser



e) Population of s_{self} at each round, in a single game against EVChooser

Figure 5.2: 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 5.2: 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%

that, even though I showed above that s_{self} and s_{EVC} have significant differences, they performed almost identically against EVChooser in terms of population and win percentages

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

My 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. I 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. I ran 10,000 games in this manner, and percentage of wins for each strategy are shown in Table **??**.

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, I 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.

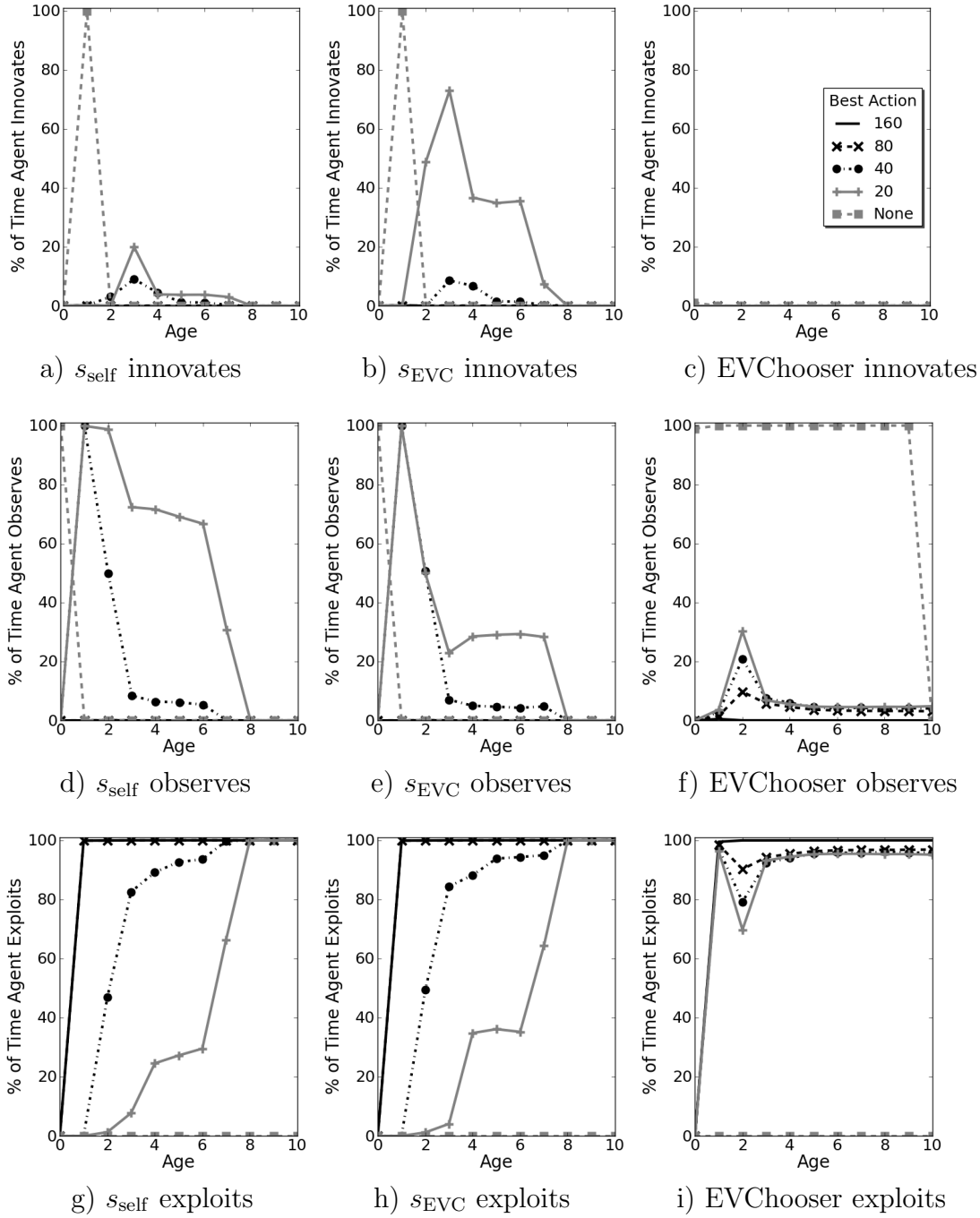
Performance Analysis of s_{self}, s_{EVC}, and EVChooser

In the experiments in Section **??**, 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, I ran two experiments to compare the performance of s_{self} and EVChooser in more detail. The first was designed to show the kinds of situations in which the two strategies chose different actions, while the second was designed to show 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, I allowed s_{self} to play against itself for five games, in an environment identical to the one used for the previous experiments in Section **??** (note that this is the same environment s_{self} was trained in). On each round, for each agent, I recorded the number of rounds the agent had lived, the value of the best action in its repertoire,[4] 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 10,000 rounds, this yielded a total of five million samples. Figures **??**(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. I then repeated this process for s_{EVC} and EVChooser, allowing each strategy to play against itself for five games and recording

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



a) s_{self} innovates b) s_{EVC} innovates c) EVChooser innovates

d) s_{self} observes e) s_{EVC} observes f) EVChooser observes

g) s_{self} exploits h) s_{EVC} exploits i) EVChooser exploits

Figure 5.3: 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.

the same data. The results for s_{EVC} and EVChooser may be found in Figures **??**(b), (e), and (h), and Figures **??**(c), (f), and (i), respectively.

The most obvious difference among the three strategies is that EVChooser almost never innovates,[5] a property it shares with the strategies that did well in the Cultaptation tournament [**?**]. 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 **??** 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

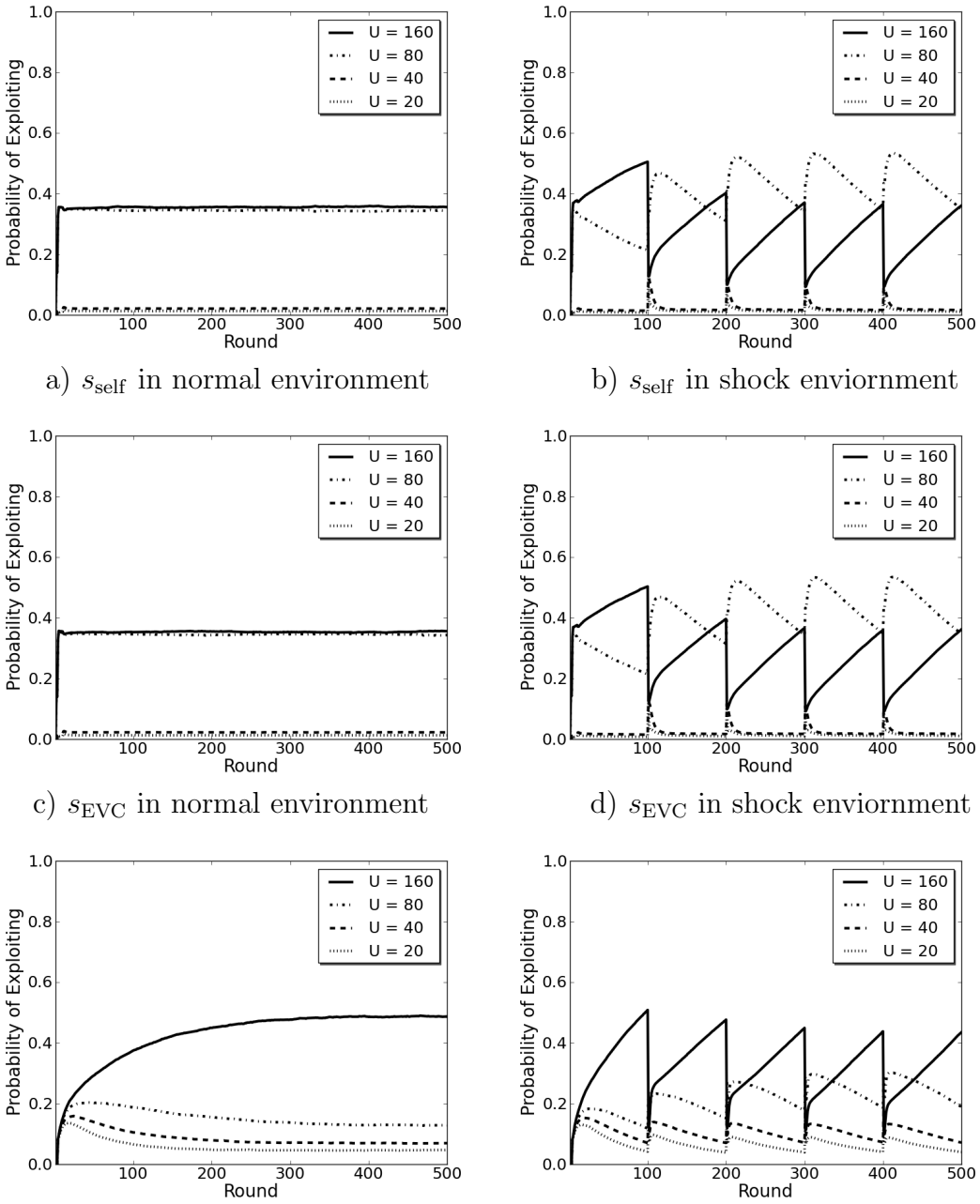
[5]EVChooser innovates 1% of the time on its first round.

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, I again played s_{self} against itself in the same environment used in the previous experiment (which I will refer to as the "normal" environment in this section), and on each round I 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, I ran 10,000 games and averaged the results for each round. I 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 **??**(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, I am also interested in seeing how they respond to structural shocks [**?**, **?**] (i.e., how quickly the strategies



a) s_{self} in normal environment

b) s_{self} in shock enviornment

c) s_{EVC} in normal environment

d) s_{EVC} in shock enviornment

e) EVChooser in normal environment

f) EVChooser in shock enviornment

Figure 5.4: 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 **??**, 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.

are able to recover when a good, widely-used action changes values). To this end, I 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 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. I 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 **??**(b), (d), and (f) respectively.

In Figure **??** 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.

Chapter 7

Conclusion

This dissertation has presented a variety of techniques for analyzing Cultaptaiton, a complex evolutionary game designed to explore the phenomenon of social learning. Furthermore, the work has demonstrated how to find strategies that are provably good, and how to analyze such strategies to draw conclusions about how good social learning strategies will operate. Thus, this work has advanced the state of the art in two ways.

First, it provides theory and analysis that support many of the empirical results found by the first Cultaptation tournament, and identifies some aspects of the tournament results that are likely due to experimental error (e.g., the lack of individual learning in any of the top-performing strategies). This helps strengthen our understanding of social learning in general, and should help inform future studies of this phenomenon.

Second, it demonstrates techniques that can be used to analyze evolutionary games that are significantly more complex than classical evolutionary games, i.e., games that have a finite number of agents, rather than an infinitely large, well-mixed population; games that last a finite, rather than infinite, number of generations; and games that allow agents to live for multiple generations and condition their actions on accumulated experience, rather than replacing the population every generation

and preventing agents from accumulating experience in the first place. One of the reasons Cultaptation is so complex is that early evolutionary models of social learning made very strong assumptions about social learning mechanics and strategies [**?**, **?**], and the conclusions drawn from studying such models generated a controversial challenge to social learning's role in evolutionary fitness that has taken decades to fully address [**?**, **?**, **?**, **?**]. As researchers in the field of evolutionary game theory continue to study more complex phenomena like social learning, it is likely that these weaker assumptions will be needed more frequently, and so techniques like the ones presented here will be necessary more often.

In summary, this dissertation has provided the following contributions:

1. Analyzing strategies' reproductive success. Given a Cultaptation game G and a set $\mathbf{S}$ of available strategies for G, the work presents a formula for approximating (to within any $\epsilon > 0$) the expected per-round utility, $\text{EPRU}(s \mid G, \mathbf{S})$, of each strategy in $\mathbf{S}$. The work shows 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. The work provides 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

S.

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.

3. Approximating symmetric Nash equilibria. The work provides CSLA, an iterative self-improvement algorithm that uses the strategy-generation algorithm in Section **??** to attempt to find 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 find a near-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. To make its algorithms fast enough for practical experimentation, the work provides a state-aggregation technique that speeds them up by an exponential factor without any loss in accuracy. The experimental results in Section **??** demonstrate the practical feasibility that this provides: in these experiments, CSLA always converged in just a few iterations.

Generalizability: The state-aggregation technique is generalizable to other evo-

lutionary games in which the utilities are Markovian.

5. Experimental results. In the experimental studies, the near-Nash equilibria produced by CSLA in any given game were all virtually identical, regardless of the starting values that were 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, the experiments show that s_{self} relies primarily on observation and exploitation, but switches quickly to innovation when a structural shock occurs, switching back to observation and exploitation once it has learned how to respond to the shock. This conflicts with the conventional wisdom [**?**, **?**] 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 [**?**] and some recent experimental results on human subjects [**?**].

6. Improvement on the best tournament strategy. While the algorithms described above are fast enough for experimentation on smaller variants of the Cultaptation game, they would be intractable for use on the more complex variant used in the Cultaptation tournament. The work shows two approaches that can extend the above results to larger environments such as these. First, it shows how the analysis and experimental results outlined above can be used to identify potential problems in the best strategy from the Cultaptation tournament (i.e. *discountmachine*). Second, it uses the formulae from the analysis to define a new strategy, *relaxedlookahead*, that avoids such weaknesses. Experimental results verify

that *relaxedlookahead* is capable of outperforming *discountmachine* in a variety of environments similar to those used in the Cultaptation tournament, and provide an in-depth analysis of the factors that allowed *relaxedlookahead* to perform better.

The analysis showed that refusing to innovate except as a bootstrapping measure (as *discountmachine* and all of the top-performing strategies from the tournament did) makes it much more difficult to recover from structural shocks, especially in environments with low probability of change. Strategies that are willing to innovate when a structural shock is detected, as *relaxedlookahead* is, are able to avoid this problem. The analysis also showed that heuristics instructing a strategy to explore with some minimum frequency, like the one used by *discountmachine*, are unnecessary, since *relaxedlookahead* exhibits emergent behavior in which it explores at intervals dependent upon the parameters of the environment, without being specifically programmed to do so.

One possible avenue for future work would be to identify computationally feasible techniques capable of approximating the Nash equilibrium strategy in these larger versions of Cultaptation, preferably with provable bounds on the difference between the performance of such techniques and the Nash equilibrium strategy. In classical games, a regret minimizing strategy[1] is typically not computationally intensive and has been shown to have performance quite close to that of a Nash strategy in several large classes of repeated games [**?**]. Therefore, it may be fruitful to

[1] in short, a regret minimizing strategy seeks to minimize the expected difference in payoff between the result of its chosen action and the best possible result if it had selected a different action.

attempt to extend the concept of regret minimization to evolutionary game theory; such an extension would need to account for the conceptual difference between payoff in classical games and fitness in evolutionary games, perhaps involving an idea of "lost fitness minimization," in which the player compares the fitness gained by its chosen action to the highest amount of fitness it would have obtained if it had selected a different action.

7. Implications. These 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 observe and 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.

Bibliography

[1] A. Bandura. *Social Learning Theory*. General Learning Press, New York, 1977.

[2] B.G. Galef and K.N. Laland. Social learning in animals: Empirical studies and theoretical models. *Bioscience*, 55:489–499, 2005.

[3] T.R. Zentall. Imitation: Definitions, evidence, and mechanisms. *Animal Cognition*, 9:335–353, 2006.

[4] 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.

[5] G. Duffy, T. Pike, and K.. Laland. Size-dependent directed social learning in nine-spined sticklebacks. *Animal Behaviour*, 78(2):371–375, August 2009.

[6] J. Kendal, L. Rendell, T. Pike, and K. Laland. Nine-spined sticklebacks deploy a hill-climbing social learning strategy. *Behavioral Ecology*, 20(2):238–244, 2009.

[7] R. Boyd and P.J. Richerson. Why does culture increase human adaptability? *Ethology and Sociobiology*, 16(2):125–143, 1995.

[8] K.N. Laland. Social learning strategies. *Learning and Behavior*, 32:4–14, 2004.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] T. Kameda and D. 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.

[14] 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.

[15] R. Boyd, M. Enquist, K. Eriksson, M. Feldman, and K. Laland. Cultaptation: Social learning tournament, 2008. `http://www.intercult.su.se/cultaptation`.

[16] 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.

[17] A. R. Rogers. Does biology constrain culture? *American Anthropologist*, 90(4):819–831, 1988.

[18] T. Wisdom and R. 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.

[19] R. Carr, E. Raboin, A. Parker, and D. Nau. Theoretical and experimental analysis of an evolutionary social-learning game. *Technical Reports from UMIACS, UMIACS-TR-2012-05*, 2012.

[20] J. C. Villaneuva. Atoms in the universe. *Universe Today*, July 2009. `http://www.universetoday.com/36302/atoms-in-the-universe/`.

[21] R. Carr, E. Raboin, A. Parker, and D. Nau. Near-optimal play in a social learning game. In *International Conference on Computational Cultural Dynamics (ICCCD), 2009*, 2009.

[22] D. Friedman. On economic applications of evolutionary game theory. *Journal of Evolutionary Economics*, 8(1):15–43, 1998.

[23] E. Guse. Expectational business cycles. Money Macro and Finance (MMF) Research Group Conference 2004 97, Money Macro and Finance Research Group, September 2004.

[24] E. Raboin, R. Carr, A. Parker, and D. Nau. Balancing innovation and exploitation in a social learning game. In *AAAI Fall Symposium on Adaptive Agents in Cultural Contexts*, November 2008.

[25] K. Leyton-Brown and Y. Shoham. *Essentials of Game Theory: A Concise Multidisciplinary Introduction*. Morgan & Claypool, 2008.

[26] R. Carr, E. Raboin, A. Parker, and D. Nau. When innovation matters: An analysis of innovation in a social learning game. In *Second International Conference on Computational Cultural Dynamics (ICCCD)*, September 2008.

[27] J. Henrich and R. McElreath. The evolution of cultural evolution. *Evolutionary Anthropology*, 12:123–135, 2003.

[28] 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.

[29] C. Papadimitriou and J. Tsitsiklis. The complexity of optimal queuing network control. *Mathematics of Operations Research*, 24(2):293–305, May 1999.

[30] S. Guha, K. Munagala, and P. 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.

[31] 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.

[32] D.E. Koulouriotis and A. Xanthopoulos. Reinforcement learning and evolutionary algorithms for non-stationary multi-armed bandit problems. *Applied Mathematics and Computation*, 196(2):913 – 922, 2008.

[33] M. Kearns, Y. Mansour, and A. Y. Ng. A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes. *MACHINE LEARNING*, 49:193–208, 2002.

[34] 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.

[35] 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.

[36] A. Plaat, J. Schaeffer, W. Pijls, and A. de Bruin. An algorithm faster than negascout and sss* in practice. In *Computer Strategy Game Programming Workshop*. 1995.

[37] T. Ibaraki. Theoretical comparision of search strategies in branch and bound. *International Journal of Computer and Information Sciences*, 5:315 344, 1976.

[38] E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, Potomac, MD, 1978.

[39] R.A. Howard. *Dynamic programming and Markov process*. MIT Press, 1960.

[40] A. Blum, M.T. Hajiaghayi, K. Ligett, and A. Roth. Regret minimization and the price of total anarchy. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 373–382. ACM, 2008.