A Competitive Approach to Game Learning

Christopher D. Rosin (contact author) and Richard K. Belew
Cognitive Computer Science Research Group
CSE Department, University of California, San Diego
La Jolla, CA 92093-0114
{crosin,rik}@cs.ucsd.edu

Abstract

Machine learning of game strategies has often depended on *competitive* methods that continually develop new strategies capable of defeating previous ones. We use a very inclusive definition of *game* and consider a framework within which a *competitive algorithm* makes repeated use of a *strategy learning* component that can learn strategies which defeat a given set of opponents. We describe game learning in terms of sets $\mathcal{H}$ and $\mathcal{X}$ of first and second player strategies, and connect the model with more familiar models of concept learning. We show the importance of the ideas of teaching set [20] and specification number [19] k in this new context. The performance of several competitive algorithms is investigated, using both worst-case and randomized strategy learning algorithms. Our central result (Theorem 4) is a competitive algorithm that solves games in a total number of strategies polynomial in $\lg(|\mathcal{H}|)$, $\lg(|\mathcal{X}|)$, and k. Its use is demonstrated, including an application in concept learning with a new kind of counterexample oracle. We conclude with a complexity analysis of game learning, and list a number of new questions arising from this work.

1 Introduction

Empirical work has been done on machine-learning systems that learn to play games by using data from their own play. Typically, a series of strategies for the game are produced during learning with strategies getting progressively stronger. Many of these game learning systems use a competitive approach that repeatedly learns new strategies capable of defeating older ones. This is the type of method that we consider. The main intuition is that it can bootstrap the level of play, from initial uninformed strategies to expert players. Examples include Samuel's classic work on checkers [15], systems using reinforcement learning [21], and using a genetic algorithm [13, 14]. The definition of "game" used in this paper is very inclusive and allows us to also consider domains other than traditional board games, such as evolving sorting networks [4], minimax controller design [16], and discrete approximations to differential games [3, 18].

To study this, our framework for game learning relies on the existence of a *strategy learning algorithm*, that is able to learn strategies which defeat a given set of opponents. The *competitive algorithm* then repeatedly uses a strategy learning algorithm to discover strong strategies for the game. We seek a competitive algorithm capable of learning perfect strategies for any game in polynomial time.

In Section 2 we give details of our model of game learning, describe its connection to familiar models of concept learning, and briefly mention some related work. Section 3 motivates the consideration of both worst-case and randomized strategy learning algorithms, and gives necessary parameters for measuring competitive algorithm performance in each case. We then examine several competitive algorithms motivated by those used in practice. Section 4 presents two simple competitive algorithms, and shows examples on which they can fail to learn perfect strategies in polynomial time. A competitive algorithm that meets our performance goals with both worst-case and randomized strategy learning algorithms is given in Section 5. Examples of its use are described, including an application to concept learning with a new kind of counterexample oracle. Section 6 explores the computational complexity of game learning, and Section 7 discusses several open problems.

2 Preliminaries

2.1 Definition of Games

A game is a function G which maps two inputs h and x (first and second player strategies) to an outcome

$G(h, x)$. The first-player strategy comes from a set of possible first-player strategies, $h \in \mathcal{H}$. Similarly, the second-player strategy comes from a set of possible second-player strategies, $x \in \mathcal{X}$. The game outcome may take on many values, usable by a strategy learning algorithm, but our framework only considers one bit of outcome information: which player was the winner. The competition is assumed to be deterministic. For simplicity, no ties are allowed.

The notation $a \succ b$ indicates that strategy a defeats strategy b. This is also extended to sets of strategies: $a \succ B$ means that $\forall b \in B, a \succ b$. $A \succ B$ is defined as $\forall b \in B, \exists a \in A$ such that $a \succ b$.[1]

2.2 Framework for Game Learning

For most of our results, it is necessary to assume that there is a perfect strategy (either for the first player or the second player) that defeats all possible opposing strategies. For notational convenience, it is always assumed to be the first player that has a perfect strategy when it is necessary to make this distinction.[2] For the most part, we consider exact learning of this perfect strategy. This simplifies matters, and seems a good way to start because there is not a clear best way to define approximate game learning.

There are two main components to the game learning systems we consider. We assume that first and second player *strategy learning algorithms* are available, that can find a strategy capable of defeating a given set A of opponents. The strategy learning algorithms are denoted L_1 for the first player, and L_2 for the second player. For L_2, no single strategy may be capable of defeating every member of A, so we require that it return a set B of strategies, such that $B \succ A$ and $|B| \leq k'$ for some constant k'.

The *competitive algorithm* is the outer loop, and uses the strategy learning algorithms to produce new strategies. In this model, the *only* methods available to the competitive algorithm for producing new strategies are the strategy learning algorithms. For example, the competitive algorithm is not allowed to use domain-specific knowledge to modify strategies. Starting points for the competitive algorithm are obtained by calling the strategy learning algorithms on the empty set. We consider several competitive algorithms, related to those used in empirical work.

[1] $\succ$ is only meaningful in the context of a particular game G and should be subscripted $\succ_G$. Whenever we use this notation, the game is clear from context, so the subscript is dropped. This is also true for several other definitions.

[2] Many board games are largely symmetric. The main reason for making a distinction between first-player and second-player strategies is the existence of a perfect strategy for one player but not the other.

As a concrete example of how this framework might be applied, consider Samuel's original work on learning evaluation functions for checkers from self-play[15]. Games were played between a fixed *Beta* player and a learning *Alpha* player. Alpha would learn from these games via Samuel's reinforcement learning algorithm; this corresponds to the strategy learning algorithm. When Alpha was finally able to defeat Beta, Beta was replaced by Alpha. The competitive algorithm uses the strategy learning algorithm to find a new (Alpha) strategy capable of defeating the current (Beta) strategy, then moves to this new strategy (makes Beta equal to Alpha).

In some empirical work on game learning, the competitive algorithm is not as explicitly defined. For example, Tesauro's backgammon learning system[21] uses reinforcement learning without explicitly checking whether new strategies defeat old ones, as a competitive algorithm would do. Given the continual improvement seen in strategies, it is likely that this condition is being met implicitly (similar results were obtained when this condition *was* checked explicitly by sampling the outcome of several games between new and old strategies[13]). So, our framework might be used to explain the performance of this system.

The separation between competitive algorithm and strategy learning algorithm seems natural because strategy learning is an optimization problem with a fixed cost function; this type of optimization has been well-studied. We focus on the power of competitive algorithms, with few restrictions on strategy learning algorithms used. This approach is what allows us to use techniques from computational learning theory.

"Time" for a competitive algorithm refers to the total number of strategies considered by it. Polynomial time for a competitive algorithm will actually be polynomial clock-time as long as the strategy learning algorithm requires polynomial clock-time.

2.3 Correspondence with Concept Learning

Our model for game learning shares similarities with the typical model of concept learning. The first-player strategy space $\mathcal{H}$ corresponds to concept learning's hypothesis space; the assumption that it has a perfect strategy is similar to assuming that the hypothesis space contains the target concept. The first-player strategy learning algorithm corresponds to a hypothesis learner, and the second-player strategy learning algorithm can be viewed as a counterexample oracle. The competitive algorithm corresponds to various learning protocols that use a hypothesis learner multiple times (for example, alternating equivalence queries and calls to an algorithm providing consistent hypotheses).

Having a hypothesis h consistent with the target on an example x corresponds to $h \succ x$. An important difference between concept learning and game learning is that concept learning is primarily concerned with discovering how the target classifies new examples, whereas game learning requires the actual "target" strategy. In this sense, game learning is a more restrictive model. Positive results from game learning should carry over to concept learning (an example of this will be seen below) and negative results from concept learning should carry over to game learning.

2.4 Related Work

A large amount of work has been done on reinforcement learning. Some results in reinforcement learning have been described in terms of game learning [7]. But, most of these results show convergence without considering learning time, or prove time bounds that are polynomial in the number of states [10]. Also, proven results usually rely on simple lookup table representations for value functions. These are not useful for complex domains in which the number of states is vast, but there have been promising recent results using certain kinds of value function approximators [9].

Several recent papers have discussed adaptive strategies for simple repeated games [11, 5]. The goal of such strategies is typically to learn enough about a particular opponent in early games to do well against it in later games. These concerns are largely orthogonal to our goal of learning strategies that are robust against a large space of opponents.

3 Competitive Algorithm Performance

Performance of competitive algorithms will always depend on $\lg(|\mathcal{H}|)$ and $\lg(|\mathcal{X}|)$. This type of dependence is familiar from concept learning, and shouldn't be prohibitive for reasonable representations of strategies. In this section, we establish necessary additional parameters for studying competitive algorithm performance.

3.1 Specification Number

For a particular game G with strategy spaces $\mathcal{H}$ and $\mathcal{X}$, define a *teaching set* T for G as a subset of $\mathcal{X}$ such that for any imperfect strategy $h \in \mathcal{H}$, $\exists x \in T$ such that $x \succ h$. Define the *specification number* k for G to be the size of the smallest teaching set. These definitions follow the corresponding ones for concept learning [19, 20].

Since we are considering exact learning, we seek bounds on learning time that are polynomial in k, as well as in $\lg(|\mathcal{H}|)$ and $\lg(|\mathcal{X}|)$. The following lemma shows that the dependence on k is necessary.

Lemma 1 *For games G with at most (a constant) c perfect strategies and with specification number k, there exists a randomized first-player strategy learning algorithm L_1 such that any competitive algorithm using it takes expected time $\Omega(k)$ to learn a perfect strategy.*

Proof: Let $L_1(A)$ choose a strategy uniformly at random from the set $\{h \in \mathcal{H} | h \succ A\}$; this choice of L_1 will become important in Section 3.3. Let T be the minimal teaching set for G; $|T| = k$. Each member of T must defeat at least one member of $\mathcal{H}$ that no other member of T defeats (otherwise there would be no reason to include it and T would not be minimal). So, for any A with $|A| < |T|$, there must be at least $k - |A|$ members of $\mathcal{H}$ not defeated by A. The probability of $L_1(A)$ returning a perfect strategy is at most $\frac{c}{k-|A|}$.[3] The probability that a competitive algorithm finds a perfect strategy by time t (for $t < k$) is at most $t\frac{c}{k-t}$, since at most t calls were made to $L_1(A)$, each with $|A| \leq t$. By time $t = \frac{k}{2c+1}$, the probability of finding a perfect strategy is at most $\frac{1}{2}$, giving a total expected time of at least $\frac{k}{4c+2}$, which is $\Omega(k)$. $\Box$

As an example, consider a class of games G^n_{hard}. G^n_{hard} has n first-player strategies, only one of which is perfect, and $n-1$ second-player strategies, each defeating only one imperfect first-player strategy. The specification number for this game is $k = n - 1$. Any competitive algorithm using the first-player strategy learning algorithm described above requires $\Omega(k) = \Omega(n)$ time to learn a perfect strategy for this game; this is not polynomial in $\lg(|\mathcal{H}|)$ and $\lg(|\mathcal{X}|)$.

This indicates that time bounds for competitive algorithms need to depend on k, as well as $\lg(|\mathcal{X}|)$ and $\lg(|\mathcal{H}|)$. For game learning to be practical, k should be fairly small; this is the case for specific examples described below.

The second-player strategy learning algorithm requires $k' \geq k$ to ensure that it can always succeed. We assume that k' is polynomial in k, $\lg(|\mathcal{H}|)$, and $\lg(|\mathcal{X}|)$.

[3] This shows the necessity of bounding the number of perfect strategies by a constant. If c were allowed to grow with k, this probability might always be large.

3.2 Worst-case Strategy Learning Algorithms

In order to demonstrate the power of competitive techniques, we would like a competitive algorithm that can solve games in time polynomial in $\lg(|\mathcal{X}|)$, $\lg(|\mathcal{H}|)$, and k with *worst-case* choices of L_1 and L_2. Unfortunately, the following lemma shows that this is not possible. First, a definition is needed:

Define a *transitive chain* of length ℓ in a game to be a sequence of pairs (h_i, X_i) $(i = 1, 2, \dots \ell)$, with $h_i \in \mathcal{H}$ and $X_i \subseteq \mathcal{X}$, $|X_i| \leq k'$, such that:

1. $\forall i > j, h_i \succ X_j$
2. $\forall i \geq j, X_i \succ \{h_j\}$

Lemma 2 *For games G with a transitive chain of length ℓ, there exist L_1 and L_2 such that any competitive algorithm using these strategy learning algorithms requires $\Omega(\ell)$ time to learn a perfect strategy.*

Proof: Assume G has a transitive chain $(h_1, X_1) \dots (h_\ell, X_\ell)$.

If L_1 and L_2 are passed the empty set, let them return h_1 and X_1, respectively.

If A contains only members from the transitive chain, let $L_1(A)$ return h_i, where $A = \{X_{i_1}, X_{i_2}, \dots X_{i_{|A|}}\}$ and $i = (\max_j i_j) + 1$. Otherwise, let L_1 return the perfect strategy.

Similarly, if B contains only members from the transitive chain, let $L_2(B)$ return X_i, where $B = \{h_{i_1}, h_{i_2}, \dots h_{i_{|B|}}\}$ and $i = (\max_j i_j) + 1$. Otherwise, let L_2 return a teaching set.

The first call to L_1 or L_2 must pass the empty set (since the competitive algorithm has no strategies available yet), so that the first strategies available to the competitive algorithm are h_1 or X_1. At least the next ℓ calls return only strategies from the transitive chain, with each successive call increasing the distance into the transitive chain by at most 1. Since none of the strategies in the transitive chain may be perfect except for the last one, the competitive algorithm cannot learn the perfect strategy in less than ℓ steps. □

As an example, consider the following class of games: $\mathcal{H}$ consists of the numbers $1 \dots n$ and $\mathcal{X}$ consists of the numbers $0 \dots n-1$. $h \succ x$ iff $h > x$. There is a perfect first player strategy and $k = 1$. But, there is a transitive chain of length n: $(0, \{0\}), (1, \{1\}), \dots (n-1, \{n-1\})$. So, with worst-case strategy learning algorithms, there is no competitive algorithm that solves these games in time polynomial in $\lg(|\mathcal{H}|)$, $\lg(|\mathcal{X}|)$, and k.

We denote by ℓ the length of the longest transitive chain in a game. Due to the above result, upper bounds for competitive algorithms using worst-case strategy learning algorithms must depend on ℓ as well as $\lg(|\mathcal{H}|)$, $\lg(|\mathcal{X}|)$, and k. In the long version of this paper, we show that there are natural classes of strategies for traditional board games that have exponentially large ℓ. But, by going beyond worst-case strategy learning algorithms, we can eliminate the need for dependence on ℓ.

3.3 Randomized Strategy Learning Algorithms

In practice, it seems unlikely that simple, natural strategy learning algorithms will be "adversarial", producing uninformative near-worst-case strategies. This is also suggested by a result in the long version of this paper, where we show that in terms of computational complexity, learning uninformative strategies is *more* difficult than our central problem of finding perfect strategies for games with polynomially large specification number.

To obtain positive results for competitive algorithms without using ℓ as a parameter, we need to restrict strategy learning algorithms in some way. Note that it would be uninformative to consider *best-case* strategy learning algorithms. Since we allow the strategy learning algorithm unlimited access to the game, any game with a solution has a first-player strategy learning algorithm that immediately produces a perfect strategy.

Another typical approach for going beyond worst-case algorithms is to use randomized learning algorithms [8]. We consider randomized strategy learning algorithms that, when passed a set A of opponents, choose from a distribution over the set of strategies (or strategy sets) that defeat A. Randomization tends to be most helpful when this distribution is required to be uniform. This is what "randomized strategy learning algorithm" will refer to below, although Theorem 4 relaxes this condition on the distribution somewhat. The importance of randomization is that it does not allow strategy learning algorithms to always choose from a few bad strategies that make the competitive algorithm fail.

Competitive algorithm time bounds will depend on $\lg(|\mathcal{H}|)$, $\lg(|\mathcal{X}|)$, and k when randomized strategy learning algorithms are used. Note that dependence on k is still necessary since the proof of Lemma 1 used a randomized strategy learning algorithm.

4 Two Simple Competitive Algorithms

Two simple competitive algorithms are described in this section that make few demands on the strategy learning algorithms. We motivate a more complex competitive algorithm by showing natural examples where the simpler competitive algorithms may fail to solve games in polynomial time, even with randomized strategy learning algorithms.

4.1 Each Defeats the Last

A simple method of learning is to have the competitive algorithm obtain an initial first-player strategy s, then find a second-player strategy t with $t \succ s$. Then, find $s' \succ t$, $t' \succ s'$, and so on. This is essentially the competitive algorithm used in Samuel's checkers learning system, and is very similar to that used in a recent backgammon learning system [13].

The main problem with this competitive algorithm is that intransitivity may exist, and a particular learner may simply keep choosing strategies in a cycle. Intransitivity has been observed when using this competitive algorithm for backgammon [13]. Even a randomized learner may get stuck in such a cycle for a long time. The following example demonstrates this.

Example 1 (Small Game Trees)
This example considers games represented by game trees. The time required by the strategy learning algorithm is polynomial in the number of nodes in the tree, so this example could be reasonably extended to familiar simple games like tic-tac-toe.

Let T_d be the complete binary tree of depth d; T_d has $n = 2^{d+1} - 1$ nodes and $l = 2^d$ leaves. Let G_d be the class of games with game tree T_d, with leaves labelled with binary outcomes in all possible 2^l ways. An outcome of 1 indicates a first-player win, while an outcome of 0 indicates a second-player win. Let $\mathcal{H}$ consist of all possible first-player strategies (all possible sets of choices at the first-player nodes). Let $\mathcal{X}$ consist of all possible second-player strategies. Note that $\lg(|\mathcal{H}|)$ and $\lg(|\mathcal{X}|)$ are $O(n)$. Each game in G_d must either be a first player win or a second player win with perfect play, so let $G'_d \subset G_d$ be the set of games in G_d that are a first player win. Since $\mathcal{H}$ contains all possible strategies for G'_d, it must contain a winning strategy for each element of G'_d.

These games have $k \leq n$. For a particular $g \in G'_d$, for each imperfect $h \in \mathcal{H}$, there must be a leaf of g with label 0 such that h may reach this leaf, given appropriate second-player play. For each leaf, there is a strategy $x \in \mathcal{X}$ that will always reach that leaf, if it is reachable given the opposing first player. Let T contain one strategy from $\mathcal{X}$ for each leaf in g labelled 0. Now, for any imperfect $h \in \mathcal{H}$, there exists $x \in T$ with $x \succ h$. So, T is a teaching set, and is of size $|T| \leq n$.

The maximum length of a transitive chain in these games is $\ell \leq n$. Each set of second-player strategies X_i in a chain must defeat h_i via a leaf l_0 unreachable by any previous second-player strategy in the chain; such a previous strategy that could reach l_0 could defeat h_i, which contradicts the definition of a transitive chain. Since each step in the transitive chain must contain a second-player strategy that can reach a leaf no previous second-player strategy could reach, the length of the longest transitive chain is bounded by the number of leaves, which is bounded by n.

A randomized strategy learning algorithm may be constructed for these games. For a particular opponent O, all that needs to be specified to defeat it is the moves along a single path through the game tree, such that O is consistent with all the choices along the path and loses the game. If O is a first player, each such path requires specification of $\lfloor \frac{d}{2} \rfloor$ moves. If O is a second player, each such path requires specification of $\lceil \frac{d}{2} \rceil$ moves. A list of all such winning paths can be constructed in time polynomial in l, since there is only one possible path for each leaf. To make the uniform choice, a random choice is made from this list and the irrelevant bits are then set to 1 with probability $\frac{1}{2}$ and to 0 with probability $\frac{1}{2}$.

The competitive algorithm described in this section will typically fail to solve such games in polynomial time using this learning algorithm. This is easy to see: the learner always sets most of the strategy bits randomly. For example, consider games where the first player must always choose the right branch to win. Considering all possible second-player responses, at least $2^{\lceil \frac{d}{2} \rceil} - 1$ bits must be correctly set to have a perfect first-player strategy, only $\lceil \frac{d}{2} \rceil$ of which will actually be specified (not set randomly) in any strategy produced by the learning algorithm. Correctly guessing the remaining bits will require time doubly exponential in d, which is exponential in n.

For this example, the competitive algorithm fails to learn a perfect strategy in time polynomial in $\lg(|\mathcal{H}|)$, $\lg(|\mathcal{X}|)$, k, and ℓ, even using randomized strategy learning algorithms.

Adding finite memory to this competitive algorithm, by always choosing a strategy that defeats the last h opponents, is of limited usefulness; it can still fail on cycles with period greater than h. The next competitive algorithm uses a memory of *all* previous second-player strategies.

4.2 Single Counterexamples

In exact concept learning, one type of model considers equivalence queries. Given a hypothesis, an equivalence query returns "yes" if the hypothesis is the target, and provides a counterexample if it is not. The concept learner queries with a hypothesis consistent with all examples seen; if its hypothesis is wrong, it chooses a new one consistent with all examples including the newly received counterexample. Translated into a game learning context, a counterexample is a second-player strategy chosen to defeat a proposed first-player strategy. The first-player strategy learning algorithm is now required to choose strategies that defeat *all* second-player strategies that have been seen during learning. This is a natural extension to the competitive algorithm in the previous subsection.

This competitive algorithm is sufficient to learn perfect strategies for the games in Example 1, in time polynomial in n using *any* strategy learning algorithms[4].

Counterexamples are not sufficient to learn all games, though. Angluin has shown that for some classes in concept learning, an adversary may choose uninformative counterexamples so that the target cannot be exactly learned in polynomial time [1]. Below, we show the existence of a game for which counterexamples do not ensure polynomial learnability, with either worst-case strategy learning algorithms or randomized strategy learning algorithms.

Example 2 (Generalized Guessing Games) Let $\mathcal{H}$ consist of all natural numbers in the range $0 \ldots n-1$. Each member of $\mathcal{X}$ is a union of at most I intervals in this range. A game g_y is defined by a particular number y in this range, and the particular choice of unions of intervals in $\mathcal{X}$. Every $x \in \mathcal{X}$ not containing y loses all games in g_y. Other than this, $h \in \mathcal{H}$ defeats x iff x contains h. To ensure that the specification number is 1, the single interval containing only y is included in $\mathcal{X}$.

As a worst-case example, assume that $\mathcal{X}$ consists of all unions of at most two intervals that exclude a single number, and the interval containing only y. The second-player strategy learning algorithm chooses the interval excluding h as a counterexample to the first-player strategy h (assuming $h \neq y$). The first-player strategy learning algorithm makes a choice uniformly at random from the intersection of all passed second-player strategies. Since each counterexample eliminates only one first-player strategy, this requires $\Omega(n)$ expected time to learn the perfect strategy. Since ℓ is $O(1)$ (the only second-player strategy defeating more than one first-player strategy is the interval containing only y), this time bound is not polynomial in $\lg(|\mathcal{H}|)$, $\lg(|\mathcal{X}|)$, k, and ℓ.

For an example where randomized strategy learning algorithms fail, let the first-player strategy learning algorithm be defined as above and let the second-player strategy learning algorithm choose uniformly from all members of $\mathcal{X}$ containing the passed first-player strategy. Let $\mathcal{X}$ contain all unions of at most two intervals that exclude only elements $i \ldots (i + \lfloor\sqrt{n}\rfloor) \bmod n$, for each i. $\mathcal{X}$ also contains the interval consisting only of y, but the probability of choosing this informative counterexample is only $\frac{1}{\lfloor\sqrt{n}\rfloor+1}$ each step of the competitive algorithm. After t steps, the probability of the first-player strategy learning algorithm choosing y is at most $\frac{1}{n-t(\lfloor\sqrt{n}\rfloor+1)}$. So, the expected time to find y is not polynomial in $\lg n$.

[4]Details are given in the long version of this paper.

5 The Covering Competitive Algorithm

5.1 Covering all First and Second Player Opponents

Counterexamples to single first-player strategies fail to provide polynomial learnability for all games. Note that the first-player strategy learning algorithm, at every step, finds a strategy that defeats all second-player strategies already seen. It seems natural to produce new second-player strategies in a similar way. Unless the specification number is 1, it may not always be possible to provide a single second-player strategy that defeats all first-player strategies already seen, but as long as the perfect first player has not yet been chosen, it is always possible to choose a set of second players of size k that covers all first players already seen.

This competitive algorithm will be called the *covering competitive algorithm.* Each step, the first-player strategy learning algorithm is called on all previous second-player strategies, then the second-player strategy learning algorithm is called on all previous first-player strategies (including the new one from the current step).

The idea of measuring coverage of prior opponents has been discussed in the context of empirical game learning systems [3].

5.2 Using Worst-case Strategy Learning Algorithms

This competitive algorithm performs as well as possible with worst-case strategy learning algorithms.

Theorem 3 *Assume a game G has maximum transitive chain length ℓ. The covering competitive algorithm learns a perfect strategy for G in at most $\ell(k'+1)$ strategies.*

This is true simply because this competitive algorithms explicitly constructs transitive chains. This gives at most ℓ steps, each of which obtains at most one first-player and k' second-player strategies. Since k' is polynomial in k, $\lg(|\mathcal{H}|)$, and $\lg(|\mathcal{X}|)$, this time bound is polynomial in the relevant parameters.

5.3 Using a Randomized Strategy Learning Algorithm

For this positive result, it is desirable to relax the uniform-choice condition for randomized strategy learning algorithms. Define the (p, q)-*randomization criterion* for sampling from a set Y, as follows. The choice must be made from a distribution D over Y, with the property that there exist constants p and q ($0 < p, q \leq 1$) such that there are at least $p|Y|$ elements of Y to which D assigns a probability of at least $\frac{q}{|Y|}$.

At each step of this competitive algorithm, denote by X the set of remaining feasible sets of second-player strategies (the sets of at most k' second-player strategies that defeat all previously chosen first-player strategies). Similarly, call the remaining set of feasible first-player strategies H.

Theorem 4 *Assume that the first-player strategy learning algorithm, at each step, makes a choice from H that satisfies the (p, q)-randomization criterion for some fixed p and q. Assume that the second-player strategy learning algorithm, on each step, chooses at most k' strategies with k' polynomial in k, $\lg(|\mathcal{H}|)$, and $\lg(|\mathcal{X}|)$. Then, the perfect strategy is found with the expected number of strategies chosen bounded by a polynomial in $\lg(|\mathcal{H}|)$, $\lg(|\mathcal{X}|)$, k, $\frac{1}{p}$, and $\frac{1}{q}$.*

Proof: The idea will be to consider sequences of m steps, during which a "sample" of m first-player strategies are chosen from the remainder of $\mathcal{H}$. By a PAC-like argument, the second-player strategies chosen at the end of the m steps to defeat this sample must eliminate a large fraction of the remaining strategies in $\mathcal{H}$. The only real complication in the proof comes from the fact that successive members of the sample may be chosen from somewhat different distributions, due to the (p, q)-randomization criterion and the fact that H may shrink over the course of the m steps.

Consider a sequence of $m = \frac{2+k'\lg(|\mathcal{X}|)}{-\lg(1-\frac{pq}{4})}$ steps of the competitive algorithm. Let H_0 be H at the start of this sequence, and let H_t be H after t steps into the sequence. Similarly, let X_0 be X at the start of this sequence, and let X_t be X t steps into the sequence. The goal is to show that a fraction of at least $\frac{p}{4}$ of H_0 will be eliminated after these m steps are completed, with high probability.

The last set of second players, x, chosen in the sequence must defeat the m first players already chosen. Consider two cases. In the first case, assume that $|H_{m-1}| \leq (1 - \frac{p}{4})|H_0|$ (note that H_{m-1} is the set of remaining first-player strategies just before x is chosen). In this case, the condition on this sequence of steps is already satisfied.

In the second case, assume that this is not true. Then, at least $1 - \frac{p}{4}$ of the original set of first players, H_0, remain (and were also present each prior step of the sequence). Consider a particular $x_{bad} \in X_0$ that is "bad": defeats less than $\frac{p}{4}$ of H_0. Each step t of the sequence of m, at most $\frac{p}{2}$ of the strategies in H_0 fail to defeat x_{bad}, or have already been eliminated. On each such step, there was a probability of at least $\frac{(2p-p^2)q}{4}$ that a first-player strategy other than one of these was chosen. This is because at least $p|H_t| - \frac{p}{2}|H_0| \geq p(1-\frac{p}{4})|H_0| - \frac{p}{2}|H_0| = \frac{2p-p^2}{4}|H_0|$ first players must: have not yet been eliminated, defeat x_{bad}, and have a probability of at least $\frac{q}{|H_t|} \geq \frac{q}{|H_0|}$ of being chosen. After m first players are chosen[5], the probability that x_{bad} remains in X_t when x is chosen is at most $(1 - \frac{(2p-p^2)q}{4})^m$. At worst, every member of X_0 is bad, so $|X_0|(1 - \frac{(2p-p^2)q}{4})^m$ is an upper bound on the probability that any bad second player remains at the last step.

Assuming that the probability of the first case is 0 (worst case), the maximum probability of failure to eliminate at least $\frac{p}{4}$ of H_0 this sequence of m steps is $|X_0|(1 - \frac{(2p-p^2)q}{4})^m$. This is at most $|X_0|(1 - \frac{pq}{4})^m$. We want

$$|X_0|(1 - \frac{pq}{4})^m < \delta$$

$$\lg(|X_0|) + m \lg(1 - \frac{pq}{4}) < \lg(\delta)$$

$$m > \frac{\lg(\frac{|X_0|}{\delta})}{-\lg(1 - \frac{pq}{4})}$$

[5]These choices are not fully independent. But the events that we consider here are independent.

Since $|X_0| \le |\mathcal{X}|^{k'}$, this will be satisfied with $\delta = \frac{1}{2}$ if

$$m > \frac{1 + k' \lg(|\mathcal{X}|)}{-\lg(1 - \frac{pq}{4})}.$$

The choice of m ensures this. Each m steps, with probability at least $\frac{1}{2}$, H will lose at least a fraction $\frac{p}{4}$ of its size; this gives an expected decrease of at least a fraction $\frac{p}{8}$. This means that the algorithm terminates in (expected) $O(\frac{\lg(|\mathcal{H}|)}{-\lg(1-\frac{p}{8})})$ sequences of m steps. Since each step adds at most k' second players (and 1 first player) to the sets, an expected total of

$$O((\frac{\lg(|\mathcal{H}|)}{-\lg(1 - \frac{p}{8})})(\frac{2 + k' \lg(|\mathcal{X}|)}{-\lg(1 - \frac{pq}{4})})(k' + 1))$$

strategies will be added to the sets before termination. This is

$$O(\frac{k'^2}{p^2 q} \lg(|\mathcal{H}|) \lg(|\mathcal{X}|)).$$

Since k' was assumed to be polynomial in k, $\lg(|\mathcal{H}|)$, and $\lg(|\mathcal{X}|)$, the covering competitive algorithm chooses an expected number of strategies polynomial in $\lg(|\mathcal{H}|)$, $\lg(|\mathcal{X}|)$, k, $\frac{1}{p}$, and $\frac{1}{q}$. □

In the long version of this paper, we show that this upper bound is nearly tight.

The second-player strategy learning algorithm has a fairly difficult task: it must find sets of strategies that cover all first-player opponents. Since this theorem allows arbitrary second-player strategy learning algorithms, we can reduce this task somewhat. Assume that an algorithm L_2' is available that takes as input a set A of first-player strategies, and returns the second-player strategy covering as many of these as possible. Since a set of size k exists that covers all of A, at least one member of this set must cover $\lceil \frac{1}{k} \rceil$ of A. Using L_2' repeatedly, a set-covering approximation can be used to create a second-player strategy learning algorithm that returns sets of size at most $k' = k \log |\mathcal{H}|$ [17].

If a randomized strategy learning algorithm is available that can find k' second-player strategies all at once, satisfying the (p, q)-randomization criterion in the space of remaining feasible sets of k' second-player strategies, then the roles in Theorem 4 may be reversed, giving the following corollary.

Corollary 5 *Assume that the second-player learning algorithm, at each step, makes a choice from X that satisfies the the (p, q)-randomization criterion. Then, the perfect strategy will be found by the covering competitive algorithm with the expected number of strategies chosen bounded by a polynomial in* $\lg(|\mathcal{H}|)$, $\lg(|\mathcal{X}|)$, k, $\frac{1}{p}$, *and* $\frac{1}{q}$.

5.4 Examples

5.4.1 Generalized Guessing Games

Use of the covering competitive algorithm gives us our desired worst-case performance on Example 2, by Theorem 3. Also, the randomized strategy learning algorithms described for it satisfy the (p, q)-randomization criterion with $p = q = 1$, allowing the covering competitive algorithm to solve the game in time polynomial in n.

5.4.2 Concept Learning

An application of game learning can be reflected back to concept learning, using a new kind of counterexample oracle.

Example 3 (Learning Teachable Concepts) Assume a concept class $\mathcal{H}$ (serving as hypothesis and target class) and a space of examples, $\mathcal{X}$. A "game" g_t is associated with each $t \in \mathcal{H}$, with $g_t(h, x)$ being 1 (victory for h) if t is consistent with h on x, and 0 otherwise. An oracle is assumed available to provide *counterexample sets*. If the specification number (of a particular g_t) is k, the oracle, when given a set of (imperfect) hypotheses H, returns a set X of at most k' examples such that for each $h \in H$, $g_t(h, x) = 0$ for some $x \in X$. Assume that k' is polynomial in k, $\lg(|\mathcal{H}|)$, and $\lg(|\mathcal{X}|)$. Each step, the learner finds a hypothesis consistent with the current set of counterexamples, and adds it to the current set of hypotheses. Then, the oracle provides a set of examples that cover the current set of hypotheses; the members of this set are added to the current set of counterexamples. The target will be learned exactly in an expected number of examples polynomial in $\lg(|\mathcal{H}|)$, $\lg(|\mathcal{X}|)$, k, $\frac{1}{p}$, and $\frac{1}{q}$ if at least one of the following two conditions holds:

1. The oracle chooses a cover of k' examples from the space of possible covers consisting of k' examples, such that this choice satisfies the (p, q)-randomization criterion.

2. The learner chooses consistent hypotheses from $\mathcal{H}$, such that this choice satisfies the (p, q)-randomization criterion.

For the second possibility, the counterexample oracle can be reduced to an oracle that returns a counterexample to as many of the current hypotheses as possible; the set covering approximation can then be used. This result shows that, with such a powerful counterexample oracle, any teachable class of hypotheses can be learned (without resorting to hypotheses outside of $\mathcal{H}$ [2]).

Note that fixing k' is crucial. If the counterexample oracle were allowed to provide as many counterexamples as passed hypotheses, it could simply provide a separate counterexample to each hypothesis. The learning model would have no more power than equivalence queries. In this sense, a small set of counterexamples is more informative than a large set of counterexamples.

5.4.3 Other Strategy Learning Algorithms

There are some games which, despite the existence of long transitive chains, have strategy learning algorithms that do not meet the (p, q)-randomization criterion, but still allow the covering competitive algorithm to learn perfect strategies in polynomial time. An example of such a strategy learning algorithm is given in the long version of this paper. It learns evaluation functions represented as conjunctions of predicates over state attributes.

6 Computational Complexity

We briefly consider the computational complexity of game learning. More details are given in the long version of this paper.

We need to define games more carefully to understand the sort of computation that can be done on them. Assume that the first and second player strategies are n_1 and n_2 bits long, and that binary game outcome is given by a boolean formula f of length polynomial in n_1 and n_2. A decision-problem version of game learning can be formulated: is the game a first-player win? That is, if game outcome is given by $f(h, x)$, decide whether $\exists h \forall x f(h, x)$. But this is simply the canonical $\Sigma_2 P$-complete decision problem $QSAT_2$ [12],[6] so game learning is $\Sigma_2 P$-complete.

The problem solved by the strategy learning algorithm is in NP (a formula containing several copies of f, with different opponents inserted, must be satisfied). Satisfying the (p, q)-randomization criterion is not much more difficult: the method of *almost uniform generation* [6] can be used to do this in randomized polynomial time with an NP oracle. This implies that games can be solved with an NP oracle in expected time polynomial in $\lg(|\mathcal{H}|)$, $\lg(|\mathcal{X}|)$, and k using the covering competitive algorithm.

Lemma 2 showed that, using our definition of a competitive algorithm, no competitive algorithm exists that solves every game in time polynomial in $\lg(|\mathcal{H}|)$, $\lg(|\mathcal{X}|)$, and k with worst-case strategy learning algorithms. It is natural to ask whether there exists *any* type of algorithm that can use worst-case strategy learning algorithms to solve any game in polynomial time. As a first step, remove the dependence on k; as described above, game learning is then $\Sigma_2 P$-complete. But, if there is an algorithm that uses worst-case strategy learning algorithms to solve games in time polynomial in $\lg(|\mathcal{H}|)$ and $\lg(|\mathcal{X}|)$, game learning would be in $\Delta_2 P$ (since the strategy learning algorithms could be replaced by an NP oracle); this would yield the unlikely conclusion that $\Sigma_2 P \subseteq \Delta_2 P$.

[6] Formally, by our definition of the game learning problem, h and x need to be padded so that the length of f is polynomial in n.

7 Open Questions and Future Directions

The above result strongly suggests that games with polynomial-time computable outcomes are not solvable in time polynomial in $\lg(|\mathcal{H}|)$ and $\lg(|\mathcal{X}|)$ by any type of competitive algorithm given worst-case strategy learning algorithms. An important open question is whether games may be solvable in time polynomial in $\lg(|\mathcal{H}|)$, $\lg(|\mathcal{X}|)$, *and* k using worst-case strategy learning algorithms, without violating constraints from computational complexity. If the answer is no, then any type of competitive algorithm will require restricted games or restricted strategy learning algorithms to succeed (as is the case for the competitive algorithms discussed in this paper). If the answer is yes, then the competitive algorithms considered in this paper are restrictive; additional means of manipulating strategies offer additional power.

The (p, q)-randomization criterion is one condition that allows the covering competitive algorithm to learn perfect strategies in polynomial time. There may be other natural restrictions on strategy learning algorithms which also allow polynomial learnability. Specific broadly applicable strategy learning algorithms are also a clear target for study; the example mentioned in Section 5.4.3 may be seen as a first step in this direction.

For complex games, it is unlikely that natural classes of quickly computable strategies will contain perfect strategies. How should approximate learning be approached in these cases? One natural idea is to seek randomized strategies that win with at least probability P_w. Since this defines a new boolean outcome for the game (whether or not the first player wins with probability at least P_w), the original framework can be used. Further work is needed to take into account the nondeterministic outcome of sampling.

8 Conclusion

We have shown the existence of a competitive algorithm that is able to successfully bootstrap its way to perfect strategies for a game under fairly general conditions. This gives some validity to the intuition behind the competitive approach to game learning. The covering competitive algorithm guarantees progress by ensuring that new strategies defeat all previous strategies. Simpler algorithms that fail to meet such a condition may be unable to make rapid progress on some games.

We have provided initial results for exact learning in the competitive approach to game learning. Future work should be able to extend results to approximate learning, explore efficient strategy learning algorithms for interesting classes of strategies, and otherwise bring the theory closer to practical applicability.

Acknowledgments

Thanks to Mark Land and Ramamohan Paturi for helpful discussions. Thanks to the ACISE program at Lawrence Livermore National Laboratory for supercomputing resources used in related empirical work.

References

[1] Angluin, D. (1990) Negative results for equivalence queries. *Machine Learning* 5:2.

[2] Bshouty, N.H., R. Cleve, S. Kannan, and C. Tamon. (1994) Oracles and queries that are sufficient for exact learning. *COLT 94.*

[3] Cliff, D. and G. Miller. (1995) Tracking the Red Queen: Measurements of Adaptive Progress in Co-evolutionary Simulations. *Third European Conference on Artificial Life.*

[4] Hillis, W.D. (1991) Co-evolving Parasites Improve Simulated Evolution as an Optimization Procedure. *Artificial Life II.*

[5] Freund, Y., M. Kearns et al. (1995) Efficient Algorithms for Learning to Play Repeated Games Against Computationally Bounded Adversaries. *FOCS 36.*

[6] Jerrum, M., L. Valiant, and V. Vazirani. (1986) Random Generation of Combinatorial Structures from a Uniform Distribution. *Theoretical Computer Science* 43:2.

[7] Littman, M. (1994) Markov Games as a Framework for Multi-agent Reinforcement Learning. *Machine Learning: Proceedings of the Eleventh International Conference.*

[8] Mass, W. (1991) On-line Learning with an Oblivious Environment and the Power of Randomization. *COLT 91.*

[9] Gordon, G.J. (1995) Stable Function Approximation in Dynamic Programming. *Proceedings of the Twelfth International Conference on Machine Learning.*

[10] Fiechter, C.-N. (1994) Efficient Reinforcement Learning. *COLT 94.*

[11] Kilian, J., K.J. Lang, and B.A. Pearlmutter. (1994) Playing the Matching-Shoulders Lob-Pass Game with Logarithmic Regret. *COLT 94.*

[12] Papadimitriou, C. (1994) *Computational Complexity.*

[13] Pollack, J., A. Blair, and M. Land. (1995) Coevolution of a Backgammon Player. Submitted to *Artificial Life V.*

[14] Rosin, C., and R. Belew. (1995) Finding Opponents Worth Beating: Methods for Competitive Co-evolution. *Proceedings of the Sixth International Conference on Genetic Algorithms.*

[15] Samuel, A. (1963) Some Studies in Machine Learning Using the Game of Checkers. *Computers and Thought.*

[16] Sebald, A.V., and J. Schlenzig. (1994) Minimax Design of Neural Net Controllers for Highly Uncertain Plants. *IEEE Transactions on Neural Networks* Jan. 1994. 5: 73-82.

[17] Chvatal, V. (1979) A greedy heuristic for the set-covering problem. *Mathematics of Operations Research* 4: 233-235.

[18] Sims, K. (1994) Evolving 3D Morphology and Behavior by Competition. *Artificial Life IV*

[19] Anthony, M., G. Brightwell, D. Cohen, and J. Shawe-Taylor. (1992) On Exact Specification by Examples. *COLT 92.*

[20] Goldman, S.A., and M.J. Kearns. (1991) On the Complexity of Teaching. *COLT 91.*

[21] Tesauro, G. (1995) Temporal Difference Learning and TD-Gammon. *CACM 38:3.*