Coevolutionary Free Lunches

David H. Wolpert, dhw@email.arc.nasa.gov
William G. Macready, wgm@email.arc.nasa.gov
NASA Ames Research Center
Moffett Field, CA, 94035

Abstract— **Recent work on the mathematical foundations of optimization has begun to uncover its rich structure. In particular, the "No Free Lunch" (NFL) theorems state that any two algorithms are equivalent when their performance is averaged across all possible problems. This highlights the need for exploiting problem-specific knowledge to achieve better than random performance. In this paper we present a general framework covering more search scenarios. In addition to the optimization scenarios addressed in the NFL results, this framework covers multi-armed bandit problems and evolution of multiple co-evolving players. As a particular instance of the latter, it covers "self-play" problems. In these problems the set of players work together to produce a champion, who then engages one or more antagonists in a subsequent multi-player game. In contrast to the traditional optimization case where the NFL results hold, we show that in self-play there are free lunches: in coevolution some algorithms have better performance than other algorithms, averaged across all possible problems. We consider the implications of these results to biology where there is no champion.**

I. INTRODUCTION

Recently, the mathematical foundations of optimization have begun to be uncovered [?], [?], [?], [?], [?], [?], [?], [?], [?]. One particular result in this work, the "No Free Lunch" (NFL) theorems, establishes the equivalent performance of all optimization algorithms when averaged across all possible problems.[1] As an example of these theorems, recent work has explicitly constructed objective functions where random search outperforms evolutionary algorithms [?]. There has also been much work extending these early results to different types of optimization (e.g. to multi-objective optimization [?]). The web site www.no-free-lunch.org offers a list of recent references.

However, all this previous work has been cast in a manner that does not cover repeated game scenarios where the "objective" or "fitness" function for one player or agent can vary based on the response of another player. In particular, the NFL theorems do not cover such scenarios. These game-like scenarios are usually called "coevolutionary" since they involve the behaviors of more than a single agent or player [?].

One important example of coevolution is "self-play," where from the system designer's perspective, the players "cooperate" to train one of them as a champion. That champion is then pitted against an antagonist in a subsequent multi-player game. The goal is to train that champion player to perform as well as possible in that subsequent game. For a checkers example see [?].

[1]More precisely, the algorithms must be compared after they have examined the same number of distinct configurations in the search space.

Early work on coevolutionary scenarios includes [?], [?], [?]. More recently, coevolution has been used for problems that on the surface appear to have no connection to a game (for an early application to sorting networks see [?]). Coevolution in these cases enables escape from poor local optima in favor of better local optima.

We will refer to all players other than the one of direct attention as that player's "opponents," even when, as in self-play, the players can be viewed as cooperating. Sometimes when discussing self-play we will refer to the specific opponent to be faced by a champion in a subsequent game — an opponent not under our control — as the champion's "antagonist."

In this paper we present a mathematical framework that covers both traditional optimization and coevolutionary scenarios. It also covers other scenarios such as multi-armed bandits. We then use that framework to explore the differences between traditional optimization and coevolution. We find dramatic differences between the traditional optimization and coevolutionary scenarios. In particular, unlike the fundamental NFL result for traditional optimization, in the self-play domain there are algorithms that are superior to other algorithms for *all* problems. However in the typical coevolutionary scenarios encountered in biology, where there is no champion, NFL still holds.

Section II summarizes the previous NFL work that we extend, and Section III motivates these extensions. Section IV presents the resultant extended NFL framework, and provides example illustrations of the framework. Section V applies the NFL extensions to self play, and Section VI demonstrates that NFL results need not apply in this case. We conclude in Section VII.

II. BACKGROUND

Motivated by the myriad heuristic approaches to combinatorial optimization a number of researchers have sought insight into how best to match optimization algorithms to problems. The importance of this task was highlighted in [?]. We review the approach taken in that paper as it forms the starting point for our coevolutionary extensions.

We consider search over a finite space X and assume that the associated space of possible "fitness" or "objective function" values Y is also finite. The sizes of the spaces are $|X|$ and $|Y|$ respectively. The space of possible fitness functions, $F = Y^X$, contains $|Y|^{|X|}$ possible mappings from X to Y. A particular mapping in F is indicated as $f \in F$. All of the results mentioned in this section be extended to the case of stochastic fitness functions specified by conditional distributions $P(y \in Y \mid x \in X)$ rather than single-valued functions from X to Y. (This is explicitly demonstrated below when we introduce the generalized version of the original NFL framework.) However for pedagogical simplicity here we restrict attention to single-valued f's. We are interested in the performance of algorithms when averaged across some distribution $P(f)$ of such single-valued fitness functions.

The formalization of algorithms used in [?] is motivated by the behavior of algorithms like genetic algorithms, simulated annealing, and tabu search. All such algorithms sample elements of the search space (i.e., select an $x \in X$), and evaluate the fitness $y = f(x) \in Y$ of that sample. New x's are selected based upon previously sampled x's and the associated fitness values. At an iteration at which a total of m distinct x's have been examined we write those x's and associated fitness values as an ordered set of m *distinct* configurations: $d_m \equiv \{(d_m^x(1), d_m^y(1)), \cdots, (d_m^x(m), d_m^y(m))\}$.[2] Configurations in d_m are ordered according to the time at which the algorithm sampled them. Thus, $d_m^x(t)$ is the t'th sampled x and $d_m^y(t) = f(d_m^x(t))$ is the associated fitness. The ordered sets of all X and Y values is indicated as d_m^x and d_m^y respectively. Algorithms are compared on the basis of the samples d_m that they generate.

It is important to note that the x's in d_m^x must all be distinct. This means that algorithms are compared only on the basis of the unique x's they have examined. This does not mean that algorithms that do revisit x's (as genetic algorithms and simulated annealing typically do) cannot be compared. Rather, it means they must be compared based on the number of distinct x's they have examined. Further discussion of this point is found in [?].

Based upon these definitions an algorithm is a (perhaps non-deterministic) mapping from a set of samples d_m to a new (i.e., not yet visited) point in the search space, $d_{m+1}^x(m+1)$. That mapping is specified by the probability distribution $P_m(d_{m+1}^x(m+1) = x|d_m)$ defined over X which gives the probability of the algorithm selecting x at time $m+1$. To ensure that search space points are not revisited we require zero probability on previously visited x's. Thus, $P_m(d_{m+1}^x(m+1) = x|d_m) = 0$ for all $x \in d_m^x$. The algorithm begins with the selection of a starting configuration as specified by an initial distribution $P_1(d_1^x(1) = x)$. An algorithm a is then a specification of the probability distributions P_1, P_2, etc. (This definition of a search algorithm was also used in [?] in the case where the mapping was assumed to be deterministic.) With every visit to a new search space element the set of samples is extended from d_m to include the new x and its fitness, i.e., $d_{m+1} = d_m \cup \{x, f(x)\}$ so that $d_{m+1}^x(m+1) = x$ and $d_{m+1}^y(m+1) = f(x)$. While covering many classes of algorithms (like simulated annealing, genetic algorithms, tabu search, etc), not all algorithms are of this type (e.g., enumerative algorithms like branch and bound). The results presented here do not necessarily apply to algorithms outside the class we consider.

The efficacy of a search algorithm is assessed with a performance measure, $\Phi(d_m^y)$, which is a function of *all* the fitness values seen by the algorithm by step m. It is important to note that this measure of performance differs from the typical concerns of computationally complexity. We are not concerned with run times or memory issues. The performance of an algorithm a after having visited m distinct x's, averaged over a class of optimization problems specified with a distribution $P(f)$, is $\mathbb{E}(\Phi|m,a) = \sum_{f \in F} \Phi(d_m^y) P(d_m^y|f,m,a) P(f)$. When $P(f)$ is uniform over any set of functions which is closed under permutations[3] then it can be shown that $P(d_m^y|m,a) = \sum_f P(d_m^y|m,a,f)P(f)$ is independent of a [?], [?], [?]. Thus, the expected performance of any pair of algorithms is equal under that average. The most general form for $P(f)$ for which NFL results remain valid is derived in [?].

[?] considers many extensions of this basic result and shows that algorithms may be distinguished once we look beyond simply average performance. Results independent of the distribution over problems $P(f)$ may also be derived [?].

Our purpose here is to extend the framework discussed above to coevolutionary settings where there is more than a single player. As we shall see, such an extension can be developed which addresses many problems of interest in both evolutionary and coevolutionary optimization. Before presenting that extended framework formally, we motivate its extensions through consideration of an idealized coevolutionary optimization problem, and the k-armed bandit.

III. Motivation

A. Self Play

We can view the NFL framework reviewed above as a "game" in which a single player is trying to determine what "move" x it should make to optimize $\Phi(d_m^y)$. As an example of another type of problem we would like to study we consider self-play. This extension involves moves of more than one player, even though there *is still a single* Φ *and* f. For example, in the case of two players the fitness function depends upon the moves of both players, indicated as $\underline{x}$ and $\overline{x}$.

To illustrate this consider a multi-stage game involving the two players [?], [?], like checkers [?]. Have the players be computer programs. In this setting $\overline{x}$ and $\underline{x}$ are the two complete computer programs that compete with each other, rather than the plays they make at any particular stage. These programs, fixed at the beginning of the game, specify each player's entire strategy of what play to make in response to what set of preceding observations. It is these programs that are of interest. In noncooperative game theory these programs are called "normal form strategies". In other applications, $\underline{x}$ might represent an algorithm to sort a list, and $\overline{x}$ a mutable set of lists to be sorted. The payoff f then reflects the ability of the algorithm to sort the lists in $\overline{x}$.

In self-play we fix attention to the payoff to one of the two players, the "champion", with the other player being the "opponent". A fitness function $f(\underline{x}, \overline{x})$ gives the reward to the champion (e.g., $+1$ for a pair of strategies in which it wins, 0 for an indeterminate or drawing pair, and -1 for a losing pair). Now concatenate the strategies of our player ($\underline{x}$) and the

[2]This set was called a trace in [?].

[3]$P(f)$ is closed under permutations if for any permutation $\sigma : X \rightarrow X$ of inputs then $P(f) = P(f \circ \sigma)$.

opponent ($\overline{x}$) into a single joint point $x = [\underline{x}, \overline{x}]$. By doing this we do not need to generalize fitness functions when we extend the NFL framework; the fitness function remains a mapping from $X = \underline{X} \times \overline{X}$ into Y. Now, however, X is the space of joint (champion, opponent) game strategies.

In the more general form of self-play this approach is extended by having several players compete in a tournament, and from the results of that tournament selecting a single best agent. That best agent constitutes the champion, who will compete against an antagonist in a subsequent game. The goal is to design the tournament to produce the champion with the best possibility of beating the antagonist. We would like to assess the efficacy of various such designs, and to see if NFL-like results also hold in this game setting.

When designing a self-play tournament there are two different choices to make. First, one must decide how the "training games" are selected, i.e., how each set of all the players' strategies for the next round of the tournament are chosen based on the results of the preceding rounds. Second, one must decide how to use the outcomes of all those games to select the champion.

As in the original NFL work, the m distinct training games and their fitnesses are indicated as $d_m \equiv \{(d^x_m(1), d^y_m(1)), \cdots (d^x_m(m), d^y_m(m))\}$. Analogously we write the probabilistic mapping that selects each new training game's strategies based on the results of the preceding ones as a set of conditional distributions. We write that set as the "algorithm" a, exactly as in the original NFL framework.

Choosing a champion is done with a function A which maps a completed sequence of training games, d_m, into a champion strategy/move. We parameterize that champion as the associated subset of all joint strategies X consistent with it. For example, say that our champion strategy takes the role of the first player in a 2-player subsequent game with a single antagonist. In other words, our champion is a choice of a (hopefully) optimal first player's strategy. So we choose that champion by selecting a particular value $\underline{x}^*$ for the strategy $\underline{x}$ of the first player in the subsequent game. Since that choice of strategy doesn't restrict the antagonist's responses, we indicate it as the subset of all $x \in X$ with $\underline{x} = \underline{x}^*$, i.e. the subset $\{(\underline{x}, \overline{x}) | \overline{x} \in \overline{X}\}$. So A maps d_m to such a subset of X. (In the more general approach of Sec. V A is allowed to map probability distributions over X, not just subsets of X.)

How do we judge the performance of the champion when we do not know how the antagonist will act? One possibility is to measure the performance of the champion against the antagonist who performs best against the champion. If the champion plays the game according to $\underline{x}^*$, then this worst case measure may be written as $\min_{\overline{x}} f(\underline{x}^*, \overline{x})$ where $\overline{x}$ ranges over all possible opponent strategies. Having defined $A(d_m)$ as above we can also write the worst case performance as $\min_{x \in A(d_m)} f(x)$. A good champion will maximize this worst possible performance. Here we see the first difference from the original single-player NFL scenario. In that original optimization setting performance is solely a function of d^y_m (the observed game outcomes), here however, the maximin criteria has an explicit dependence on the fitness function f. As we shall see, it is this dependence which will give rise to free lunches in which there can be *a priori* differences between algorithms.

Other possible means of quantifying the performance of the champion are possible, and in some cases preferable. Subtleties in evaluating the performance of game-playing strategies are considered in [?], [?], [?]. In this work we concentrate on the maximin measure, but we expect that if the performance measure depends explicitly on f then generically NFL type results will not hold.

B. *Bandit Problems*

The k-armed bandit problem is simple, but captures much of the essence of the critical exploration/exploitation tradeoff inherent in optimization. In this problem an agent is faced with repeatedly choosing between k stochastic processes having different means. With each selection (either process 1, process 2, $\cdots$, process k) the agent receives a reward stochastically sampled from the process it chooses. The agent's goal is to maximize the total reward collected over m selections. One simple strategy is to sample each process n times for a total of kn training points, and for the remaining $m - kn$ time steps to sample that process which has the higher empirical mean based on the n points sampled from each process. An algorithm of this type was proposed (erroneously) as justification for the schema theorem of genetic algorithms [?], [?].

In order to allow NFL-like analyses to apply to algorithms for bandit problems we must generalize the notion of a fitness function. In this case the fitness of any given x value ($x = i$ for selecting process i) is not deterministic, but stochastic, given by sampling the associated process. To capture this we extend the definition "fitness function" from a $X \rightarrow Y$ mapping to mapping from $X \rightarrow Z$, where Z is a space capturing probabilistic models. This is illustrated below.

IV. General Framework

As we have seen from these two examples to increase the scope of NFL-like analyses we need to make two slight extensions. Firstly, we must broaden the definition of performance measures to allow for dependence on f, and secondly, we need to generalize fitness functions to allow for non-determinism. The resultant framework is closely related to the one used in the very first work on NFL, preceding its application to the problem of search, namely NFL for supervised machine learning [?], [?], [?].

A. *Formal framework specification*

We assume two spaces, X and Z. To guide the intuition, a typical scenario might have $x \in X$ be the joint strategy followed by our players, and $z \in Z$ be one of the possible probability distributions over some space of possible rewards to the champion.

In addition to X and Z, we also have a *fitness function*

$$f : X \rightarrow Z. \tag{1}$$

In the example where z is a probability distribution over rewards, f can be viewed as the specification of an x-conditioned probability distribution of rewards. In particular, single-valued fitness functions are special cases of such an f, where each $f(x)$ — each x-conditioned probability distribution — is a delta function about some associated reward value. Different such f give different single-valued mappings from x to rewards. The introduction of Z into the framework is what allows for noisy payoffs, and to allow it to cover bandit problems.

We have a total of m time-steps, and represent the samples generated through those time-steps as

$$d_m \equiv (d^x_m, d^z_m) \equiv \left(\{d^x_m(t)\}^m_{t=1},\ \{d^z_m(t)\}^m_{t=1}\right).$$

As in classic NFL each $d^x_m(t)$ is a particular $x \in X$. Each $d^z_m(t)$ is a (perhaps stochastic) function of $f\big(d^x_m(t)\big)$. For example, say z's — values of $f(x)$ — are probability distributions over reward values. Then $d^z_m(t)$ could consist of the full distribution $f\big(d^x(t)\big)$. Alternatively, it could consist of a moment of that distribution, or even a random sample of it. In general, we allow the function specifying $d^z_m(t)$ to vary with t. However that freedom will not be exploited here. Accordingly, we will leave that function implicit, to minimize the notation. As shorthand we will write $d(t)$ to mean the pair $\big(d^x_m(t), d^z_m(t)\big)$.

A *search algorithm*, a, is an initial distribution $P_1(d^x_m(1))$ of the initially selected point $d^x_m(1) \in X$, together with a set of $m-1$ separate conditional distributions $P_t(d^x_m(t) \mid d_{t-1})$ for $t = 2, \ldots, m$. Such an algorithm specifies which x to next choose, based on the samples observed so far, for any time-step t. As is usual, we assume that the next x has not been previously seen. This is reflected as an implicit restriction on the conditional distributions $P_t(d^x_m(t) \mid d_{t-1})$.

Finally, we have a (potentially vector-valued) *cost function*, $C(d_m, f)$, which is used to assess the performance of the algorithm. Often our goal is to find the a that will maximize $\mathbb{E}(C)$ for a particular choice of the mapping forming the $d^z_m(t)$'s from the $f(d^x_m(t))$'s. This expectation $\mathbb{E}(C)$ is formed by averaging over any stochasticity in the mapping from f's to associated $d^z_m(t)$'s. It also averages over those fitness functions f consistent (in the sense of Bayes' theorem) with the observed samples d_m. (See below for examples.)

The NFL theorems concern averages over all f of quantities depending on C. For those theorems to hold — for f-averages of C to be independent of the search algorithm — it is crucial that for fixed d_m, C does not depend on f. When that independence is relaxed, the NFL theorems need not hold. As we have seen such relaxation occurs in self-play; it is how one can have free lunches in self-play.

B. Examples of the framework

Example 1: One example of this generalized framework is the scenario considered in the original NFL theorems. There we can identify Z with a distribution over Y where Y is a subset of $\mathbb{R}$ (for convenience we take X and Y countable). For single-valued fitness functions, as remarked above, such distributions must be delta functions. In this case the implicit mapping from $f(d^x_m(t))$ to the associated $d^z_m(t)$ is given simply by evaluating the real value f has at $d^x_m(t)$. As an alternative formulation, for such fitness functions we can instead define $z \in Z$ to be the same as Y. (Recall that Z need not be a space of probability distributions; that's only the choice of Z used for illustrative purposes.) In the more general version of the original NFL scenario Z is a non-delta function over Y, and the mapping $f(d^x_m(t))$ to the associated $d^z_m(t)$ is given by forming a sample of $f(d^x_m(t))$.

In the scenario of the original NFL theorems a does not allow revisits. In addition we take $C(d_m, f) = \Phi(d_m)$ (recall the definition of the performance measure Φ in section II). As already noted, for NFL to hold it is critical that the cost function does not depend on f. It is also crucial that the search algorithm a not allow revisits. Both apply to the formulation given here. Accordingly, the NFL theorems generically apply to scenarios which can be cast as an instance of this example.

Example 2: While the variables are interpreted differently (e.g., x is now a joint strategy, not a single sample point), the formal specification of self-play in terms of our extended framework is almost identical to that of the original (noisy fitness function) NFL scenario. The only formal difference between the scenarios arises in the choice of C.

In self-play we use the set of repeated games, together with any other relevant information we have (e.g., how the game against the antagonist might differ from the games heretofore), to choose the champion strategy to be used in the subsequent game against the antagonist. As we have seen, this dependence is given by a function $A(d_m)$ mapping d_m to a subset of X. Since it measures performance against the antagonist, C must involve this specification of the champion.

Formally, C uses A to determine the quality of the search algorithm that generated d_m as follows:

$$C(d_m, f) = \min_{x \in A(d_m)} \mathbb{E}(f). \tag{2}$$

where $\mathbb{E}(f)$ is the expected value of the distribution of rewards our champion receives for a joint strategy with the antagonist given by x:

$$\mathbb{E}(f) = \sum_{y \in Y} y P_f(y \mid x) = \sum_{y \in Y} y[[f(x)](y)] \tag{3}$$

where $[[f(x)](y)]$ is the distribution $f(x)$ evaluated at y.

This cost function is the worst possible payoff to the champion. There are several things to note about it. First, it still applies if the number of players in any game is greater than 2 (the number of players just determines the dimensionality of x, and the form of the function A). Also A arises nowhere in our formulation of self-play but in this specification of C. Finally, note that the C of self-play depends on f.

Say we have a 2-player self-play scenario and the antagonist has no care for any goal other than hurting our champion. Say that the antagonist is also omnipotent (or at least very lucky), and chooses the $\bar{x}$ which achieves its goal. Then the expected reward to the champion is given by Eq. (2). Obvious variants of this setup replace the worst-case nature of C with some alternative, have A be stochastic, etc.

Whatever variant we choose, typically our goal in self-play is to choose a and/or A so as to maximize $\mathbb{E}(C)$, with the expectation now extending to average all possible f. The fact that C depends on f means that NFL need not apply though.

Examples of this are presented below, in Sections V, V-B, and VI.

Example 3: Another example is the k-armed bandit problem introduced for optimization by Holland [?], and analyzed thoroughly in [?]. The scenario for that problem is identical to that for the NFL results, except that there are no constraints that the search algorithm not revisit previously sampled points, $Y = \mathbb{R}$, and every z is a Gaussian. The fact that revisits are allowed (since typically $m > k$) means that NFL need not apply.

Example 4: In the general biological coevolution scenario [?], [?], [?] there is a set of "players" who change their strategies from one game to the next, just like in self-play. Unlike in self-play though, each player has an associated frequency in the population of all players, and that frequency also varies through the succession of games. This means that the two scenarios are quite different when formulated with our framework. Moreover, the formulation for the general coevolution scenario involves definitions of Z, f, etc., that would appear counter-intuitive if we were to interpret them the same way we do in self-play.

We formulate the general coevolution scenario with our framework by having a set of N agents (or players, or cultures, or lineages of genomes, or lineages of genes, etc), just like in self-play. Their strategy/move spaces are written X_i, as in self-play. Now however X is extended beyond the current joint strategy to include the joint "population frequency" value of those strategies. Formally, we write

$$X = (X_1, u_1) \times \cdots \times (X_N, u_N), \qquad (4)$$

and interpret each $x_i \in X_i$ as a strategy of i and each $u_i \in \mathbb{R}$ as a frequency with which i occurs in the overall population of all players.[4]

To be more precise, we interpret $x_i(t)$ as i's current strategy. However we interpret $u_i(t)$ as i's *previous* population frequency, i.e., the population frequency, at the preceding timestep, of the strategy that i followed then. In other words, we interpret the u_i component of $d^x_m(t)$ as the population frequency at timestep $t-1$ of the strategy followed by agent i then, a strategy given by the X_i component of $d^x_m(t-1)$. So the information concerning each agent i is "staggered" across pairs of successive timesteps. This is done so that a can give the sequence of joint population frequencies that accompanies the sequence of joint strategies, as described below.

When i is a single agent, this choice of X accomodates learning in i by allowing the strategy of i, $x_i \in X_i$, to change from one timestep to the next. When i is a "lineage of a gene" it is not the strategy (i.e., the gene) that changes from one timestep to the next, but the associated frequency of that strategy in the population. This too is accomodated in our choice of X; changes in X from one time-step to the next can involve changes in the joint-frequency without any changes in the joint-strategy. More generally, our formulation allows both kinds of changes to occur simultaneously . In addition, mutation, e.g., modification of the gene, can be captured with this framework. This is done by having some i's that at certain times have 0 population frequency, but then stochastically jump to non-0 frequency, representing a new agent that is a mutant of an old one.

Have each z be a probability distribution over the possible *current* population frequencies of the agents. So given our definition of X, we interpret f as a map taking the previous joint population frequency, together with the current joint strategy of the agents, into a probability distribution over the possible current joint population frequencies of the agents

As an example, in evolutionary game theory, the joint strategy of the agents at any given t determines the change in each one's population frequency in that time-step. Accordingly, in the replicator dynamics of evolutionary game theory, f takes a joint strategy $x_1 \times \ldots x_N$ and the values of all agents' previous population frequencies, and based on that determines the new value of each agent's population frequency. More precisely, $d^z_m(t)$ is a sample of that distribution $f(d^x_m(t))$.

In this general coevolution scenario, our choice for a, which produces $d^x_m(t+1)$ from $d^x_m(t)$, plays two roles. These correspond to its updates of the strategy components of $d^x_m(t)$ and of the population frequency components of $d^x_m(t)$, respectively. More precisely, one of the things a does is update the population frequencies from those of the previous timestep $t-1$ (which are stored in $d^x_m(t)$) to the ones given by $d^z_m(t)$. This means directly incorporating those population frequencies into the $\{u_i\}$ components of $d^x_m(t+1)$. The other thing a does, as before, is determine the joint strategy $[x_1, \ldots, x_N]$ for time $t+1$. At the risk of abusing notation, as in self-play we can write the generation of the new strategy of each agent i by using a (potentially stochastic and/or time-varying) function written a_i. In sum then, an application of a to a common d_t is given by the simultaneous operation of all those N distinct a_i on d_t, as well as the transfer of the joint population frequency from $d^z(t)$. The result of these two processes is $d^x(t+1)$.

Note that the new joint strategy produced by a may depend on the previous time-step's population frequencies, in general. As an example, this corresponds to sexual reproduction in which mating choices are stochastic, so that how likely agent i is to mate with agent j depends on the population frequencies of agents i and j.[5] However in the simplest version of evolutionary game theory, the joint strategy is actually constant in time, with the only thing that varies in time being the population frequencies, updated in f. If the agents are identified with distinct genomes, then in this version of evolutionary game theory reproduction is parthenogenic.

The choice of C depends on what one wishes to know about a sequence d_m and f. Typical analyses performed in population biology and associated fields have C be a vector with N components, each component j depending only on the associated $d_t(j)$. As an example, typically in evolutionary game theory each component is j's population frequency at

[4]Typically $\sum_i u_i = 1$ of course, though we have no need to explicitly require this here. Indeed, the formalism allows the u_i not to be population frequencies, but rather integer-valued population counts.

[5]Obvious elaborations of the framework allow X to include relative rewards between agents in the preceding round, as well as the associated population frequencies. This elaboration would allow mate selection to be based on current differential fitness between candidate mates, as well as their overall frequency in the population.

$t-1$.

In general in such biological analyses there is no notion of a champion being produced by the search and subsequently pitted against an antagonist in a "bake-off." (Famously, evolution is not teleological.) Accordingly, unlike in self-play, there is no particular significance to results for alternative choices of C that depend on f in such analyses. This means that so long as we make the approximation, reasonable in real biological systems, that x's are never revisited, all of the requirements of Example 1 are met. This means that NFL applies.

Some authors have promoted the use of the general coevolution scenario as a means of designing an entity to perform well, rather than as a tool for analyzing how a system happens to develop. In general, whether or not NFL applies to such a use will depend on the details of the design problem.

For example, say the problem is to design a value y that maximizes a provided function $g(y)$, e.g., design a biological organ that can function as an optical sensor. Then, even if we are in the general coevolutionary scenario of interacting populations, we can still cast the problem as an instance of the choices of Z, f, etc., of Example 1. In particular, for our design problem C does not involve any "subsequent game against an antagonist", and C is independent of f. So the NFL theorems hold; the extra details of the dynamics introduced by coevolution don't affect the validity of those theorems, which is independent of such details. On the other hand, say the problem is to design an organism that is likely to avoid extinction (i.e., have a non-zero population frequency) in the years after a major change to the ecosystem. For this problem the coevolution scenario is a variant of self-play; the "years after the major change to the ecosystem" constitute the "subsequent game against an antagonist". In this situation NFL may not hold.

There are other ways one can express the general coevolution scenario in our framework, i.e., other choices for the roles of f, a, etc. The advantage of the one used here is how it formally separates the different aspects of the problem. f plays the role of the laws of Nature which map joint strategies and population frequencies to new population frequencies (e.g., the replicator dynamics). All variability in how one might update strategies — cross-over, mutation, etc. — are instead encapsulated in a. In particular, if one wishes to compare two such update schemes, without knowing anything about f ahead of time or being able to modify it, that means comparing two different a's, while f is fixed and not something we can have any knowledge about.

V. Application to Self-play

In section III-A we introduced a model of self-play. In the remainder of this paper we show how free lunches may arise in this setting, and quantify the *a priori* differences between certain self-play algorithms.

To summarize self play, we recall that agents (game strategies) are paired against each other in a (perhaps stochastically formed) sequence to generate a set of 2-player games. After m distinct training games between an agent and its opponents, the agent enters a competition. Performance of the agent is measured with a payoff function. The payoff function to the agent when it plays strategy $\underline{x}$ and it's opponent plays $\overline{x}$ is written as $f(x)$ where $x = (\underline{x}, \overline{x})$ is the joint strategy. We make no assumption about the structure of strategies except that they are finite.

We define the payoff for the agent playing strategy $\underline{x}$ independent of an opponent's reply, $g(\underline{x})$, as the least payoff over all possible opponent responses: $g(\underline{x}) \equiv \min_{\overline{x}} f(\underline{x}, \overline{x})$. With this criterion, the best strategy an agent can play is that strategy which maximizes g (a maximin criterion) so that its performance in competition (over all possible opponents) will be as good as possible. We are not interested in search strategies just across the agent, but more generally across the joint strategies of the agent and its opponents. (Note that whether that opponent varies or not is irrelevant, since we are setting its strategies.) The ultimate goal is to maximize the agents performance g.

We make one important observation. In general, using a random pairing strategy in the training phase will not result in a training set that can be used to guarantee that any particular strategy in the competition is better than the worst possible strategy. The only way to ensure an outcome guaranteed to be better than the worst possible is to exhaustively explore all possible responses to strategy $\underline{x}$, and then determine that the worst value of f for all such joint strategies is better than the worst value for some other strategy, $\underline{x}'$. To do this requires that m is greater than the total number of possible strategies available to the opponent, but even for very large m unless all possible opponent responses have been explored we can not make any such guarantees.

Pursuing this observation further, consider the situation where we know (perhaps through exhaustive enumeration of opponent responses) that the worst possible payoff for some strategy $\underline{x}$ is $g(\underline{x})$ and that another joint strategy $x' = (\underline{x}', \overline{x}')$ with $\underline{x} \neq \underline{x}'$ results in a payoff $f(x') < g(\underline{x})$. In this case there is no need to explore other opponent responses to $\underline{x}'$ since it must be that $g(\underline{x}') < g(\underline{x})$, i.e., $\underline{x}'$ is maximin inferior to $\underline{x}$. Thus, in designing an algorithm to search for good strategies, any algorithm that avoids searching regions that are known to be maximin inferior (as above) will be more efficient than one that searches these regions (e.g., random search). This applies for all g, and so the smarter algorithm will have an average performance greater than the dumb algorithm. Roughly speaking, this result avoids NFL implications because varying uniformly over all g does not vary uniformly over all possible f, which are the functions that ultimately determine performance.

In the following sections we develop this observation further.

A. Definitions

We introduce a few definitions to explore our observation. We assume that there are $\underline{l}$ strategies available to an agent, and label these using $\underline{X} \equiv [1, \cdots, \underline{l}]$. For each such strategy we assume the opponent may choose from one of $\overline{l}(\underline{x})$ possible strategies forming the space $\overline{X}(\underline{x})$.[6] Consequently, the size of the joint strategy space is $|X| = \sum_{\underline{x}=1}^{\underline{l}} \overline{l}(\underline{x})$. For simplicity we take $\overline{X}(\underline{x})$ to be independent of $\underline{x}$ so that $\overline{X} = [1, \cdots, \overline{l}]$ and $|X| = \underline{l}\overline{l}$. If the training period consists of m distinct joint strategies, even with m as large as $|X| - \underline{l}$, we cannot guarantee that the agent will not choose the worst possible strategy in the competition as the worst possible strategy could be the opponent response that was left unexplored for each of the $\underline{l}$ possible strategies.

As always, a sample of configurations (here configurations are joint strategies) is a sample of distinct points from the input space X, and their corresponding fitness values. For simplicity we assume that fitness payoffs are a deterministic function of joint strategies. Thus, rather than the more general output space Z, we assume payoff values lie in a finite totally ordered space Y. Consequently, the fitness function is the mapping $f : X \to Y$ where $X = \underline{X} \times \overline{X}$ is the space of joint strategies. As in the general framework, a sample of size m is represented as

$$d_m = \left\{ \left(d_m^x(1); d_m^y(1)\right), \cdots, \left(d_m^x(m); d_m^y(m)\right) \right\}$$

where $d_m^x(t) = \{d_m^{\underline{x}}(t), d_m^{\overline{x}}(t)\}$ and $d_m^y(t) = f(d_m^{\underline{x}}(t), d_m^{\overline{x}}(t))$ and $t \in [1, \cdots, m]$ labels the samples taken. In the above definition $d_m^{\underline{x}}(t)$ is the t'th strategy adopted by the agent, $d_m^{\overline{x}}(t)$ is the opponent response, and $d_m^y(t)$ is the corresponding payoff. As usual, we assume that no joint configurations are revisited, and that an algorithm a defined exactly as in the classic NFL case is used to generate sample sets d_m. A particular coevolutionary optimization task is specified by defining the payoff function that is to be maximized. As discussed in [?], a class of problems is defined by specifying a probability density $P(f)$ over the space of possible payoff functions. As long as both X and Y are finite (as they are in any computer implementation) this is conceptually straightforward.

There is an additional consideration in the coevolutionary setting, namely the decision of which strategy to apply in the competition based upon the results of the training samples. In the framework we have outlined this choice is buried in the performance measure through the function $A(d_m)$. Recall that $A(d_m)$ is a function which, given a sample of games and outcomes, returns a probability distribution over a subset of X. In the case where $A(d_m)$ is deterministic and selects the champion strategy $\underline{x}^*$ based on d_m then the subset output by A is $\{(\underline{x}^*, \overline{x}) \mid \overline{x} \in \text{possible opponent responses to } \underline{x}^*\}$.

If A is deterministic the natural empirical measure of the performance of the search algorithm a obtained during training is

$$\hat{C} = \min_{x \in A(d_m) \bigcap d_m^x} f(x).$$

Though we shall not pursue it here, it is a simple matter to allow for non-deterministic A. In such cases $A(d_m)$ might stochastically define an optimal strategy $\underline{x}^*$ through specification of d_m-dependent probability density $\rho(\underline{x}^* | d_m)$ over $\underline{X}$. In this situation, performance could be defined as the weighted average

$$\sum_{\underline{x}^* \in \underline{X}} \rho(\underline{x}^* | d_m) \min_{\overline{x} \in \overline{X}(\underline{x}^*)} f(\underline{x}^*, \overline{x}),$$

where the min over $\overline{x}$ is over possible opponent responses to $\underline{x}^*$. It is also straightforward to include a distribution over opponent responses if that were known.

To summarize, search algorithms are defined exactly as in classical NFL, but performance measures are extended to depend both on f and A. The best a for a particular f and A are those that maximize C.

The original version of NFL (for traditional optimization) defines the performance differently because there is no opponent. In the simplest case, the performance of a (recall that there is no champion-selecting procedure) might be measured as $C = \max_{t \in [1,m]} d_m^y(t)$. One traditional NFL result states that the average performance of any pair of algorithms is identical, or formally, $\sum_f P(C|f, m, a)$ is independent of a.[7] A natural extension of this result considers a non-uniform average over fitness functions. In this case the quantity of interest is $\sum_f P(C|f, m, a)P(f)$ where $P(f)$ weights different fitness functions.

A result akin to this one in the self-play setting would state that the unform average $\sum_f P(C \mid f, m, a, A)$ is independent of a. However, as we have seen informally, such a result cannot hold in general since a search process with an a that exhausts an opponent's repertoire of strategies has better guarantees than other search processes. A formal proof of this statement is presented in section VI.

B. An Exhaustive Example

Before proving the existence of free lunches we provide a small exhaustive example to illustrate our definitions, and to show explicitly why we expect free lunches to exist. Consider the case where the player has two possible strategies, i.e., $\underline{X} = \{1, 2\}$, the opponent has two responses for each of these strategies, i.e., $\overline{X} = \{1, 2\}$, and there are two possible fitness values, $Y = \{1/2, 1\}$. The 16 possible functions are listed in Table I. We see that the maximin criteria we employ gives a biased distribution over possible performance measures: 9/16 of the functions have $g = [1/2 \quad 1/2]$, 3/16 have $g = [1/2 \quad 1]$, 3/16 have $g = [1 \quad 1/2]$, and 1/16 have $g = [1 \quad 1]$ where $g = [g(\underline{x} = 1) \quad g(\underline{x} = 2)]$.

If we consider a particular population, say $d_2 = \{(1, 2; 1/2), (2, 2; 1)\}$, the payoff functions that are consistent with this population are $f_9, f_{10}, f_{13}, f_{14}$ and the corresponding distribution over g functions is $\delta(g - [1/2 \ 1/2])/2 + \delta(g - [1/2 \ 1])/2$. Given that any population will give a biased sample over g functions, it may not surprising that there are free lunches. We expect that an algorithm which is able to exploit this biased sample would perform uniformly better than another algorithm which does not exploit the biased sample of g's. In the next section we prove the existence of free lunches by constructing such a pair of algorithms.

[6]Note that the space of opponent strategies varies with $\underline{x}$. This is the typical situation in applications to games with complex rules (e.g., checkers).

[7]Actually far more can be said, and the reader is encouraged to consult [?] for details.

$(\underline{x},\overline{x})$	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	$\boldsymbol{f_9}$	$\boldsymbol{f_{10}}$	f_{11}	f_{12}	$\boldsymbol{f_{13}}$	$\boldsymbol{f_{14}}$	f_{15}	f_{16}
(1, 1)	1/2	1	1/2	1	1/2	1	1/2	1	1/2	1	1/2	1	1/2	1	1/2	1
(1, 2)	1/2	1/2	1	1	1/2	1/2	1	1	1/2	1/2	1	1	1/2	1/2	1	1
(2, 1)	1/2	1/2	1/2	1/2	1	1	1	1	1/2	1/2	1/2	1/2	1	1	1	1
(2, 2)	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1	1	1	1	1	1	1	1
$\underline{x}$	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	g_9	g_{10}	g_{11}	g_{12}	g_{13}	g_{14}	g_{15}	g_{16}
1	1/2	1/2	1/2	1	1/2	1/2	1/2	1	1/2	1/2	1/2	1	1/2	1/2	1/2	1
2	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1	1	1	1

TABLE I

EXHAUSTIVE ENUMERATION OF ALL POSSIBLE FUNCTIONS $f(\underline{x},\overline{x})$ AND $g(\underline{x}) = \min_{\overline{x}} f(\underline{x},\overline{x})$ FOR $\underline{X} = \{1,2\}$, $\overline{X} = \{1,2\}$, AND $Y = \{1/2, 1\}$. THE PAYOFF FUNCTIONS LABELED IN BOLD ARE THOSE CONSISTENT WITH THE POPULATION $d_2 = \{(1,2;1/2),(2,2;1)\}$.

VI. Construction of Free Lunches

In this section a proof is presented that there are free lunches for self-play by constructing a pair of search algorithms such that one explicitly has performance equal to or better than the other for all possible payoff functions f. We normalize the possible Y values so that they are equal to $1/|Y|, 2/|Y|, \cdots, 1$. Thus, regardless of how Y values are assigned by the fitness function, our measure gives the fraction of possible fitness values having lesser or equal fitness, and thus forms a sort of normalized ranking.

As discussed earlier, we assume that all $\underline{l}$ agent strategies offer the same number of possible opponent responses, $\overline{l}$. We consider algorithms that explore $m = \underline{l}$ distinct joint samples. Agent strategies are labeled by $\underline{x} \in \{1, \cdots \underline{l}\}$ and opponent responses are labeled by $\overline{x} \in \{1, \cdots \overline{l}\}$. For simplicity we take $\underline{l} = \overline{l}$.

In the following section we consider three different algorithms and show different expected performance for all of them. For those not interested in the details of the derivation of the performances a summary of results appears at the end of the section.

A. Algorithms Having Different Expected Performance

Algorithm a_1 explores the joint strategies $(1,1), \cdots, (1,m)$ and algorithm a_2 explores the joint strategies $(1,1), \cdots, (m,1)$, i.e., a_1 exhausts opponent responses to $\underline{x} = 1$ while a_2 only samples one opponent response to each of it's m possible strategies. For the champion-selection rule, $A(d_m)$, we apply the Bayes optimal rule: select the strategy $\underline{x}$ that has the highest expected $g(\underline{x})$ when averaged uniformly over payoff functions consistent with the observed population.

To start, we determine the expected performance of an algorithm that does not have the benefit of knowing any opponent responses. In this case we average the performance, $g(\underline{x})$, for any element $\underline{x}$, over all $|Y|^{\underline{l}\overline{l}}$ functions.[8] We note that for any given agent strategy $\underline{x}$, the $|Y|^{\underline{l}\overline{l}}$ possible function values at the joint strategies $(\underline{x}, \cdot)$ are replicated $|Y|^{\underline{l}\overline{l}}/|Y|^{\overline{l}} = |Y|^{\overline{l}(\underline{l}-1)}$ times. The number of times that a $g(\underline{x})$ value of $1 - i/|Y|$ is attained in the first $|Y|^{\overline{l}}$ distinct values is $(i+1)^{\overline{l}} - i^{\overline{l}}$. Thus the average $g(\underline{x})$ value, which we denote $\langle g \rangle$, is

$$\langle g \rangle = \sum_{i=0}^{|Y|-1} \left(1 - \frac{i}{|Y|}\right) n_{\overline{l}}(i).$$

[8]Recall that $|X| = \underline{l}\overline{l}$.

where $n_{\overline{l}}(i) = \left[(i+1)/|Y|\right]^{\overline{l}} - \left[i/|Y|\right]^{\overline{l}}$. This average value is obtained for all strategies $\underline{x}$. In the continuum limit where $|Y| \to \infty$ the expected value of g is simply

$$\langle g \rangle = 1/(1 + \overline{l}).$$

This serves as a baseline for comparison; any algorithm that samples some opponent responses has to do better than this.

Next we consider the algorithm a_1, which exhaustively explores all opponent responses to $\underline{x} = 1$. Because $m = \overline{l}$ there are $|Y|^{\overline{l}}$ possible d_m that this algorithm might see. For each of these sample sets, d_m, we need to determine $g(1)$, and the average g values for each of the other strategies $\underline{x} \neq 1$. This average is taken over the $|Y|^{\overline{l}(\underline{l}-1)}$ functions that are consistent with d_m. Of course we have $g(1) = \min d_m^y$ and the expected $g(\underline{x})$ value for $\underline{x} \neq 1$ are all equal to $\langle g \rangle$ (since we have no samples from any strategies $\underline{x} \neq 1$). Since the champion-choosing rule maximizes the expected value of g the expected performance of a_1 for this sample set is $\max(\min d_m^y, \langle g \rangle)$. Averaged over all functions the expected performance of a_1 is

$$\langle g \rangle_1 = \frac{1}{|Y|^{\overline{l}}} \sum_{d_m^y} \max\left(\min d_m^y, \langle g \rangle\right)$$

where the sum is over all $|Y|^{\overline{l}}$ possible samples. Converting the sum over all samples into a sum over the minimum value of the population we find

$$\begin{aligned}\langle g \rangle_1 &= \sum_{i=0}^{|Y|-1} \max\left(1 - \frac{i}{|Y|}, \langle g \rangle\right) n_{\overline{l}}(i) \\ &= \sum_{i=0}^{\lfloor |Y|(1-\langle g \rangle) \rfloor} \left(1 - \frac{i}{|Y|}\right) n_{\overline{l}}(i) + \langle g \rangle \sum_{i=\lceil |Y|(1-\langle g \rangle) \rceil}^{|Y|-1} n_{\overline{l}}(i).\end{aligned}$$

If we define $i_g \equiv \lceil |Y|(1 - \langle g \rangle) \rceil$ then we obtain

$$\langle g \rangle_1 = \sum_{i=0}^{i_g - 1} \left(1 - \frac{i}{|Y|}\right) n_{\overline{l}}(i) + \langle g \rangle \left\{1 - \left(\frac{i_g}{|Y|}\right)^{\overline{l}}\right\}.$$

In the continuum limit we have

$$\begin{aligned}\langle g \rangle_1 &= (1 - \langle g \rangle)^{\overline{l}} \frac{1 + \langle g \rangle \overline{l}}{1 + \overline{l}} + \langle g \rangle \left(1 - (1 - \langle g \rangle)^{\overline{l}}\right) \\ &= \frac{1}{1 + \overline{l}} \left[1 + \left(\frac{\overline{l}}{1 + \overline{l}}\right)^{1 + \overline{l}}\right]\end{aligned}$$

where we have recalled the expected value $\langle g \rangle = 1/(1 + \overline{l})$. We note that as $\overline{l} \to \infty$ the performance of algorithm a_1 is $(1 + e^{-1})$ times that of $\langle g \rangle$.

The analysis of algorithm a_2 is slightly more complex. In this case each game occurs at a different $\underline{x}$. For any given observed set of samples the optimal strategy for the agent is to choose that $\underline{x}^*$ which has the largest fitness observed in the population. With this insight, we observe that when summing over all functions, there are $|Y|^{\bar{l}-1}$ possible completions to $\max d^y_m$ for the remaining $\bar{l}-1$ unobserved responses to $\underline{x}^*$. We must take the minimum over these possible completions to determine the expected value of g. Thus, the expected payoff for algorithm a_2 when averaging over all functions is

$$\langle g\rangle_2 = \frac{1}{|Y|^{\bar{l}}}\sum_{d^y_m}\sum_{i=0}^{|Y|-1}\min\Big(\max d^y_m, 1-\frac{i}{|Y|}\Big)n_{\bar{l}-1}(i).$$

We proceed in the same fashion as above by defining[9] $i_d \equiv |Y|(1-\max d^y_m)$ (which depends on d^y_m) so that

$$\begin{aligned}\langle g\rangle_2 &= \frac{1}{|Y|^{\bar{l}}}\sum_{d^y_m}\Big[\max d^y_m\sum_{i=0}^{i_d-1}n_{\bar{l}-1}(i)+\sum_{i=i_d}^{|Y|-1}\frac{|Y|-i}{|Y|}n_{\bar{l}-1}(i)\Big]\\ &= \frac{1}{|Y|^{\bar{l}}}\sum_{d^y_m}\Big[\max d^y_m\Big(\frac{i_{d^y_m}}{|Y|}\Big)^{\bar{l}-1}+\sum_{i=i_{d^y_m}}^{|Y|-1}\frac{|Y|-i}{|Y|}n_{\bar{l}-1}(i)\Big].\end{aligned}$$

The sum over populations is now tackled by converting it to a sum over the $|Y|$ possible values of $\max d^y_m$. The number of sequences of length $\bar{l}$ having maximum value j is $j^{\bar{l}}-(j-1)^{\bar{l}}$. Moreover, if $\max d^y_m = j/|Y|$ then $i_d = |Y|-j$ and so

$$\langle g\rangle_2 = \sum_{j=1}^{|Y|}\Big[\frac{j}{|Y|}\Big\{1-\frac{j}{|Y|}\Big\}^{\bar{l}-1}+\sum_{i=|Y|-j}^{|Y|-1}\frac{|Y|-i}{|Y|}n_{\bar{l}-1}(i)\Big]n_{\bar{l}}(j-1)$$

The continuum limit in this case is found as

$$\begin{aligned}\langle g\rangle_2 &= \bar{l}\int_0^1 dy_j\ y_j^{\bar{l}-1}\Big\{y_j(1-y_j)^{\bar{l}-1}+\\ &\qquad (\bar{l}-1)\int_{1-y_j}^1 dy\ (1-y)y^{\bar{l}-2}\Big\}\\ &= \int_0^1 dy_j\ y_j^{\bar{l}}(1-y_j)^{\bar{l}-1}+\int_0^1 dy_j\ y_j^{\bar{l}-1}\\ &\quad -\int_0^1 dy_j\ y_j^{\bar{l}-1}(1-y_j)^{\bar{l}-1}\\ &= B(\bar{l}+1,\bar{l})+1/\bar{l}-B(\bar{l},\bar{l})\end{aligned}$$

where $B(x,y)$ is the beta function defined by $B(x,y) = \Gamma(x)\Gamma(y)/\Gamma(x+y)$. For large $\bar{l}$ the Beta functions almost cancel and the expected performance for a_2 varies as $1/\bar{l}$, which is only slightly better than the performance of the algorithm that does not have access to any training data.

Summary of results:

For reference we summarize these results, and the conditions under which the results have been derived. We have considered a two player game where the player and opponent each have $\bar{l}$ possible strategies available to them. Training algorithms sample $m = \bar{l}$ distinct games and their fitnesses. Fitness values lie uniformly between 0 and 1, and measure the normalized ranking so that, e.g., the configuration having fitness 1/2 is fitter than half of all possible fitnesses.

Performance is measured by the maximin criterion (i.e., the worst-case performance of the strategy against an opponent), and averaged over all possible fitness functions. Three algorithms were considered: random search which random selects $\bar{l}$ distinct training games; algorithm a_1 which applies a single given strategy and determines the opponents best response to that strategy; and algorithm a_2 which samples a single opponent response to all its $\bar{l}$ possible strategies. In all cases the champion strategy is selected with the Bayes optimal rule which chooses the strategy having the highest expected performance given the observed data. The expected performance in each of these algorithms is:

$$\begin{aligned}\text{Random:}&\quad \frac{1}{1+\bar{l}}\\ a_1:&\quad \frac{1}{1+\bar{l}}\Big[1+\Big(\frac{\bar{l}}{1+\bar{l}}\Big)^{1+\bar{l}}\Big]\\ a_2:&\quad B(\bar{l}+1,\bar{l})+1/\bar{l}-B(\bar{l},\bar{l})\end{aligned}$$

where $B(x,y) = \Gamma(x)\Gamma(y)/\Gamma(x+y)$.

Figure 1 plots the expected performance of a_1, a_2, and random search as a function of $\bar{l}$ (recall that $m = \underline{l} = \bar{l}$). Algorithm a_1 outperforms algorithm a_2 on average for all values of $\bar{l}$.

B. Performance Difference

Though a_1 outperforms a_2 on average, it is interesting to determine the fraction of functions where a_1 will perform no worse than a_2. This fraction is given by $|Y|^{-\underline{l}\bar{l}}\sum_f\theta\big(\text{perf}_1(f)-\text{perf}_2(f)\big)$ where $\text{perf}_1(f)$ is the performance of algorithm a_1 on payoff function f, $\text{perf}_2(f)$ is the performance of algorithm a_2 on the same f, and θ is a step function defined as $\theta(x)=1$ if $x\geq 0$ and $\theta(x)=0$ otherwise. The Bayes optimal payoff for a_1 for any given payoff function f is[10]

$$\text{perf}_1(f) = \begin{cases}\min_{\overline{x}} f(1,\overline{x}) & \text{if } \min_{\overline{x}} f(1,\overline{x}) > \langle g\rangle\\ \min_{\overline{x}} f(2,\overline{x}) & \text{otherwise}\end{cases}.$$

Similarly, the performance of algorithm a_2 is given by

$$\text{perf}_2(f) = \min_{\overline{x}} f(\underline{x}^*_2,\overline{x})$$

where $\underline{x}^*_2$ is the strategy having the highest fitness observed in the sample games d_m.

To determine the performance of the algorithms for any given f we divide f into its relevant and irrelevant components as follows:

$$\begin{aligned}\frac{j_1}{|Y|} &\equiv f(1,1),\quad \frac{j_2}{|Y|} \equiv f(2,1)\\ \frac{k_1}{|Y|} \equiv \min_{\overline{x}}\{f(1,\overline{x})|\overline{x}\neq 1\},&\quad \frac{k_2}{|Y|} \equiv \min_{\overline{x}}\{f(2,\overline{x})|\overline{x}\neq 1\}\\ \frac{n}{|Y|} &\equiv \max_{\underline{x}}\{f(\underline{x},1)|\underline{x}\neq 1,2\},\\ \frac{p}{|Y|} &\equiv \min_{\overline{x}}\{f(\underline{x}^*_2,\overline{x})|\underline{x}^*_2\neq 1,2,\overline{x}\neq 1\}\end{aligned}$$

[9]There is no need to take the ceiling because i_d is automatically an integer.

[10]We have arbitrarily assumed that a_1 will select strategy 2 if it does not select strategy 1. This choice has no bearing on the result.

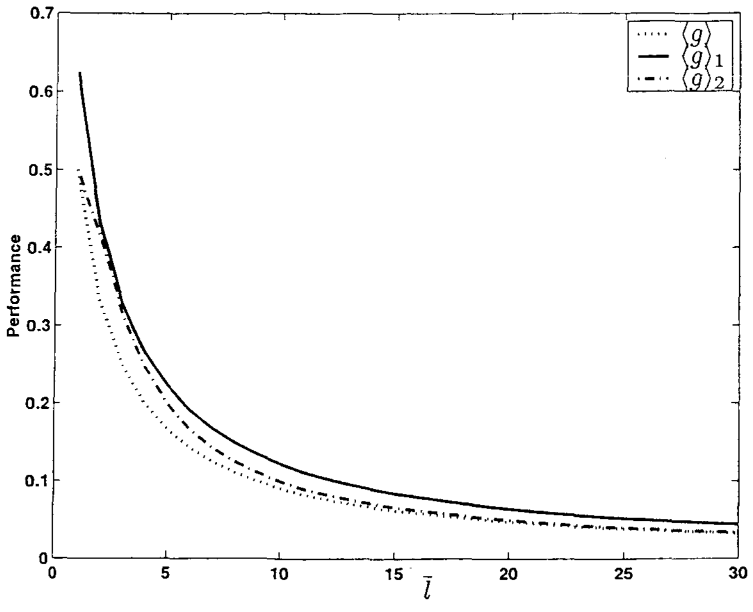

Fig. 1. Expected performance of algorithm a_1 (indicated as $\langle g\rangle_1$), which exhaustively enumerates the opponents response to a particular strategy, and algorithm a_2 (indicated as $\langle g\rangle_2$), which samples only one opponent response to each strategy. For comparison, we also plot $\langle g\rangle$, which is the expected performance of an algorithm that does no sampling of opponent responses.

In the definition of p, $\underline{x}_2^*$ is the strategy chosen by a_2; if a_2 does not choose strategy $\underline{x}_2^* = 1$ or 2, the specific value of $\underline{x}_2^*$ is irrelevant. Given these definitions, the performances of the two algorithms are

$$\text{perf}_1(f) = \frac{1}{|Y|}\begin{cases}\min(j_1,k_1) & \text{if } \min(j_1,k_1) > |Y|\langle g\rangle \\ \min(j_2,k_2) & \text{otherwise}\end{cases}$$

and

$$\text{perf}_2(f) = \frac{1}{|Y|}\begin{cases}\min(j_1,k_1) & \text{if } \max(j_1,j_2,n) = j_1 \\ \min(j_2,k_2) & \text{if } \max(j_1,j_2,n) = j_2 \\ \min(n,p) & \text{otherwise}\end{cases}$$

respectively.

In summing the above expressions over f we replace the sum over f with a sum over j_1, j_2, k_1, k_2, n, and p using the appropriate multiplicities. The resulting sums are then converted to integrals in the continuum limit and evaluated by Monte Carlo. Details are presented in Appendix A.

The results are shown in Figure 2, which plots the fraction of functions for which $\text{perf}_1 \geq \text{perf}_2$. This plot was generated using 10^7 Monte Carlo samples per $\bar{l}$ value.

C. Other Champion-Selection Criteria

We have shown the existence of free lunches for self-play by constructing a pair of algorithms with differing search rules a_1 and a_2, but with the same champion-selecting rule (select the strategy with the highest expected $g(\underline{x})$), and showed different performance. Unsurprisingly, we can construct algorithms with different expected performance which have the same search rules, but which have different champion-selecting rules. In this section we provide a simple example of such a pair of algorithms. This should help demonstrate that free lunches are a rather common occurrence in self-play settings.

Each process of the pair we construct use the same search rule a (it is not important in the present context what a is), but different deterministic champion-selecting rules $\underline{A}$.[11] In both cases a Bayesian estimate based on uniform $P(f)$ and the d_m at hand is made of the expected value of $g(\underline{x}) = \min_{\bar{x}} f(\underline{x}, \bar{x})$ for each $\underline{x}$. Since we strive to maximize the worst possible payoff from f, the optimal champion-selection rule selects the strategy that maximizes this expected value while the worst champion-selection rule selects the strategy that minimizes this value. More formally, if $\mathbb{E}(C|d_m, a, \underline{A})$ differs for the two choices of $\underline{A}$, always being higher for one of them, then $\mathbb{E}(C|m, a, \underline{A}) = \sum_{d_m} P(d_m|a)\mathbb{E}(C|d_m, \underline{A})$ differs for the two $\underline{A}$. In turn,

$$\begin{aligned}\mathbb{E}(C|m,a,\underline{A}) &= \sum_{f,C}[C \times P(C \mid f,m,a,\underline{A}) \times P(f)] \\ &\propto \sum_{f,C}[C \times P(C|f,m,a,\underline{A})]\end{aligned}$$

for the uniform prior $P(f)$. Since this differs for the two $\underline{A}$, so must $\sum_f P(C \mid f, m, a, \underline{A})$.

Let $\tilde{g}(\underline{x})$ be a random variable representing the value of $g(\underline{x})$ conditioned on d_m and $\underline{x}$, i.e., it equals the worst possible payoff (to the agent) after the agent applies strategy $\underline{x}$ and the opponent replies. In the example of section V-B we have $\mathbb{E}\tilde{g}(1) = 1/2$ and $\mathbb{E}\tilde{g}(2) = 3/4$

To determine the expected value of $\tilde{g}(\underline{x})$ we need to know $P(\tilde{g}(\underline{x}) \mid \underline{x}, d_m) = \sum_f P(\tilde{g}(\underline{x}) \mid \underline{x}, d_m, f)P(f)$ for uniform $P(f)$. Of the entire population d_m only the subset sampled at $\underline{x}$ is relevant. We assume that there are $k(\underline{x}, d_m) \leq m$ such values.[12] Since we are concerned with the worst possible

[11]The notation $\underline{A}$ is meant to be suggestive of the fact that $\underline{A}(d_m)$ is the $\underline{x}$ (first) component common to all joint configurations in $A(d_m)$.

[12]Of course, we must also have $k(\underline{x}, d_m) \leq \bar{l}(\underline{x})$ for all populations d_m.

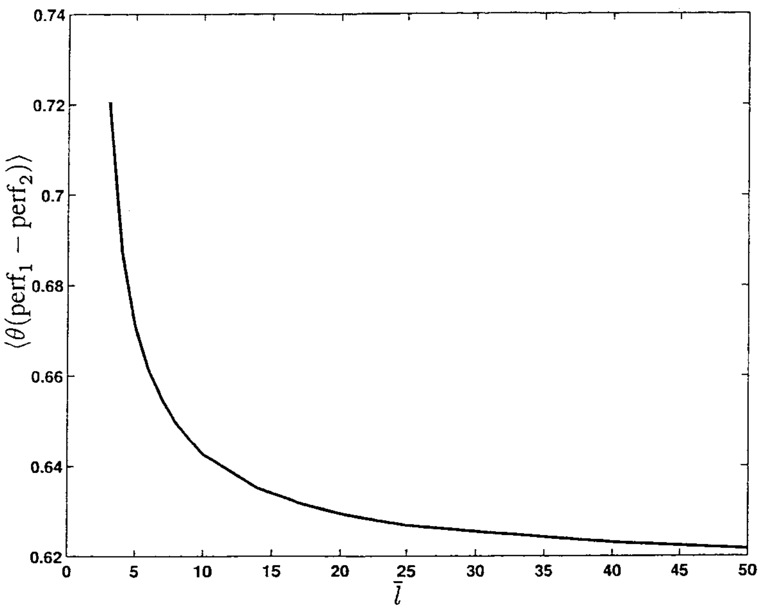

Fig. 2. The fraction of functions in the continuum limit where $\mathrm{perf}_1 \geq \mathrm{perf}_2$ The figure was generated with 10^7 Monte Carlo samples of the integrand for each value of $\bar{l}$.

opponent response let $r(\underline{x}, d_m)$ be the minimal Y value obtained over the $k(\underline{x}, d_m)$ responses to $\underline{x}$, i.e. $r(\underline{x}, d_m) = \min_{\overline{x}\in d_m^{\overline{x}}} d_m^y(\underline{x}, \overline{x})$. Since payoff values are normalized to lie between 0 and 1, $0 < r(\underline{x}, d_m) \leq 1$. Given $k(\underline{x}, d_m)$ and $r(\underline{x}, d_m)$, $P(\tilde{g} \mid \underline{x}, d_m)$ is independent of $\underline{x}$ and d_m and so we indicate the desired probability as $\pi_{k,r}(\tilde{g})$.

In appendix B we derive the probability $\pi_{k,r}$ in the case where all Y values are distinct (we do so because this results in a particularly simple expression for the expected value of $\tilde{g}$) and in the case where Y values are not forced to be distinct. From these densities we the expected value of $\tilde{g}(\underline{x})$ can be determined. In the case where Y values are not forced to be distinct there is no closed form for the expectation. However, in the continuum limit where $|Y| \to \infty$ we find (see appendix C)

$$\mathbb{E}(\tilde{g}(\underline{x}) \mid \underline{x}, d_m) = \frac{1 - (1 - r(\underline{x}, d_m))^{\bar{l}(\underline{x}) - k(\underline{x}, d_m) + 1}}{\bar{l}(\underline{x}) - k(\underline{x}, d_m) + 1}. \quad (5)$$

where we have explicitly noted that both k and r depend both on the strategy $\underline{x}$ as well as the training population d_m. As shorthand we define $C_m(\underline{x}) \equiv \mathbb{E}(\tilde{g}(\underline{x}) \mid \underline{x}, d_m)$.

The best strategy given the training population is the deterministic choice $\underline{A}_{\text{best}}(d_m) = \arg\max_{\underline{x}} C_m(\underline{x})$ and the worst strategy is $\underline{A}_{\text{worst}}(d_m) = \arg\min_{\underline{x}} C_m(\underline{x})$. In the example of section V-B with the population of size 2, $\underline{A}_{\text{best}}(d_2) = 2$ and $\underline{A}_{\text{worst}}(d_2) = 1$.

As long as $C_m(\underline{x})$ is not constant (which will usually be the case since the r values will differ) the performances of the two champion-selecting rules will differ, and the expected performance of $\underline{A}_{\text{best}}$ will be superior.

D. Better Training Algorithms

In the previous sections we constructed Bayes-optimal algorithms in limited settings by using specially constructed deterministic rules a and $\underline{A}$. This alone is sufficient to demonstrate the availability of free lunches in self-play contexts. However, we can build on these insights to construct even better (and even worse) algorithms by also determining (at least partially) the Bayes-optimal search rule, $(a, \underline{A})$, that builds out the training set, and selects the champion strategy. That analysis would parallel the approach taken in [?] used to study bandit problems. and would further increase the performance gap between the (best, worst) pair of algorithms.

E. The Role of Opponent "Intelligence"

All results thus far have been driven by measuring performance based on $g(x) = \arg\min_{\overline{x}} f(\underline{x}, \overline{x})$. This is a very pessimistic measure as it assumes that the agent's opponent is omniscient, and will employ the strategy most detrimental to the agent. If the opponent is not omniscient and cannot determine $\overline{x}^* = \arg\min_{\overline{x}} f(\underline{x}, \overline{x})$, how does this affect the availability of free lunches?

Perhaps the simplest way to quantify the intelligence of the opponent is through the fraction, α, of payoff values known to the opponent. The opponent will use these known values to estimate its optimal strategy $\overline{x}^*$. The $\alpha = 1$ limit corresponds to maximal intelligence where the opponent can always determine $\overline{x}^*$ and, as we have seen, gives free lunches. In the $\alpha = 0$ limit the opponent can only make random replies, and so that the expected performance of the agent will be the average over the opponent's possible responses.

One way to approach this problem is to build the opponent's bounded intelligence into the agent's payoff function g and proceed as we did in the omniscient case. If $|X|$ is the number of joint strategies, then there are $\binom{|X|}{\alpha|X|}$ possible subsets of joint strategies of size $\alpha|X|$.[13] We indicate the list of possible subsets as $\mathcal{S}(X, \alpha|X|)$, and a particular subset by $\mathcal{S}_i \in \mathcal{S}$. For

[13]We assume that α is an integral multiple of $1/|X|$.

this particular subset, $\overline{x}^*$ is estimated by selecting the best response out of the S_i payoff values known to the opponent. Of course, it may be the case that there are no samples in S_i having the agent's strategy $\underline{x}$ and in that case the opponent can only select a random response. In this case the agent will obtain the average payoff $\sum_{\overline{x}} f(\underline{x},\overline{x})/l(\underline{x})$. If we assume that all subsets of size $\alpha|X|$ are equally likely, then the agent's payoff function against an opponent with bounded intelligence is given by

$$g^{\alpha}(\underline{x}) = \binom{|X|}{\alpha|X|}^{-1} \sum_{S_i \in \mathcal{S}(X,\alpha|X|)} \arg\min_{(\underline{x},\overline{x})\in S_i} f(\underline{x},\overline{x}).$$

This generalization reduces to the previously assumed g in the maximally intelligent $\alpha = 1$ case. In Table II the functions $g^{1/4}$, $g^{2/4}$, $g^{3/4}$, and $g^{4/4}$ are listed for the example of section V-B. As expected the payoff to the agent increases with decreasing α (a less intelligent opponent). However, we also observe that for the same population, d_2, the average $[g(\underline{x}=1)\ g(\underline{x}=2)]$ values are $[5/8 \quad 7/8]$ for $\alpha = 1/4$, $[29/48 \quad 41/48]^{\top}$ for $\alpha = 2/4$, $[9/16 \quad 13/16]^{\top}$ for $\alpha = 3/4$, and $[1/2 \quad 3/4]^{\top}$ for $\alpha = 4/4$. For this population, d_2 $(a, \underline{A}_{\text{best}})$ continues to beat $(a, \underline{A}_{\text{worst}})$ by the same amount independent of α.

VII. Conclusions

We have introduced a general framework for analyzing NFL issues in a variety of contexts. When applied to self-play we have proven the existence of pairs of algorithms in which one is superior for all possible joint payoff functions f. This result stands in marked contrast to similar analyses for optimization in non-self-play settings. Basically, the result arises because under a maximin criteria the sum over all payoff functions f is not equivalent to a sum over all functions $\min_{\overline{x}} f(\cdot,\overline{x})$. We have shown that for simple algorithms we can calculate expected performance over all possible payoff functions and in some cases determine the fraction of functions where one algorithm outperforms another. On the other hand, we have also shown that for the more general biological coevolutionary settings, where there is no sense of a "champion" like there is in self-play, NFL still applies.

Clearly we have only begun an analysis of coevolutionary and self-play optimization. Many of the same questions posed in the traditional optimization setting can be asked in this more general setting. Such endeavors may be particularly rewarding at this time given the current interest in the use of game theory and self-play for multi-agent systems [?].

Appendix

A. Performance Comparison

In this appendix we evaluate the fraction of functions for which a_1 performs better or equal to algorithm a_2 where a_1 and a_2 are defined as in Section VI-B.

The function $\theta(\text{perf}_1(f) - \text{perf}_2(f))$ is equal to 1 if

$$cd_1 + e_1cd_2 + e_2c\overline{d}_1\overline{d}_2 + \overline{e}_1\overline{c}d_1 + \overline{c}d_2 + e_3\overline{c}\overline{d}_1\overline{d}_2$$

j_1	1
j_2	1
k_1	$(\lvert Y\rvert - k_1 + 1)^{\bar{l}-1} - (\lvert Y\rvert - k_1)^{\bar{l}-1}$
k_2	$(\lvert Y\rvert - k_2 + 1)^{\bar{l}-1} - (\lvert Y\rvert - k_2)^{\bar{l}-1}$
n	$n^{\bar{l}-2} - (n-1)^{\bar{l}-2}$
p	$(\lvert Y\rvert - p + 1)^{\bar{l}-1} - (\lvert Y\rvert - p)^{\bar{l}-1}$

TABLE III
MULTIPLICITIES OCCURRING WHEN CONVERTING THE SUM OVER f TO A SUM OVER THE ALLOWED VALUES OF j_1, j_2, k_1, k_2, l, AND p.

where $c = (\min(j_1,k_1) > |Y|\langle g\rangle)$, $d_1 = (\max(j_1,j_2,n) = j_1)$, $d_2 = (\max(j_1,j_2,n) = j_2)$, $e_1 = (\min(j_1,k_1) \geq \min(j_2,k_2))$, $e_2 = (\min(j_1,k_1) \geq \min(n,p))$, $e_3 = (\min(j_2,k_2) \geq \min(n,p))$. In the above Boolean expression we have used the condensed notation $ab \equiv a \wedge b$, $a + b \equiv a \vee b$, and $\overline{a} = \neg a$. It is convenient to factor the Boolean expression as

$$c\big(d_1 + e_1d_2 + e_2\overline{d}_1\overline{d}_2\big) + \overline{c}\big(\overline{e}_1d_1 + d_2 + e_3\overline{d}_1\overline{d}_2\big).$$

To give the fraction of functions where a_1 performs better than a_2 this expression is to be summed over j_1, j_2, k_1, k_2, n, and p with appropriate multiplicities. The multiplicities are given in Table III.

In the continuum limit this sum becomes the integral

$$\int_0^1 dj_1 \int_0^1 dj_2 \int_0^1 dk_1\, P(k_1) \int_0^1 dk_2\, P(k_2) \int_0^1 dn\, P(n) \times$$
$$\int_0^1 dp\, P(p) \big\{ c(d_1 + e_1d_2 + e_2\overline{d}_1\overline{d}_2) + \overline{c}(\overline{e}_1d_1 + d_2 + e_3\overline{d}_1\overline{d}_2) \big\}$$

where $P(k_1) = (\bar{l}-1)(1-k_1)^{\bar{l}-2}$, $P(k_2) = (\bar{l}-1)(1-k_2)^{\bar{l}-2}$, $P(n) = (\bar{l}-2)n^{\bar{l}-3}$, $P(p) = (\bar{l}-1)(1-p)^{\bar{l}-2}$, and condition c is modified to $\min(j_1,k_1) > \langle g\rangle$. Though this integral is difficult to evaluate analytically, it is straightforward to evaluate by Monte Carlo importance sampling of $(j_1, j_2, k_1, k_2, n, p)$ using the respective probability distributions. Samples from $P(u) = q(1-u)^{q-1}$ are obtained by sampling values v from $U(0,1)$ and transforming so that $u = 1 - v^{1/q}$; samples from $P(w) = qw^{q-1}$ are obtained via $w = v^{1/q}$.

B. Determination of $\pi_{k,r}(\tilde{g})$: distinct Y

To determine $\pi_{k,r}(\tilde{g})$ we first consider the case where all Y values are distinct and then consider the possibility of duplicate Y values. Though we only present the non-distinct case in the main text we derive the distinct Y case here because we can obtain a closed-form expression for the probability and because it serves as a simpler introduction to the case of non-distinct Y.

To derive the result we generalize from a concrete example. Consider the case where $|Y| = 10$, $\bar{l}(\underline{x}) = 5$, and $k = 3$. A particular instantiation is presented in Figure 3. In this case $r = 4/10$, which is not the true minimum for responses to $\underline{x}$. The probability that r is the true minimum is simply $k/\bar{l}(\underline{x})$. If r is not the true minimum then $P(\tilde{g}|d_m)$ is found as follows. $P(\tilde{g} = 1/10|d_m)$ is the fraction of functions

α	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}
1/4	$\frac{1}{2},\frac{1}{2}$	$\frac{3}{4},\frac{1}{2}$	$\frac{3}{4},\frac{1}{2}$	$1,\frac{1}{2}$	$\frac{1}{2},\frac{3}{4}$	$\frac{3}{4},\frac{3}{4}$	$\frac{3}{4},\frac{3}{4}$	$1,\frac{3}{4}$	$\frac{1}{2},\frac{3}{4}$	$\frac{3}{4},\frac{3}{4}$	$\frac{3}{4},\frac{3}{4}$	$1,\frac{3}{4}$	$\frac{1}{2},1$	$\frac{3}{4},1$	$\frac{3}{4},1$	1,1
2/4	$\frac{1}{2},\frac{1}{2}$	$\frac{17}{24},\frac{1}{2}$	$\frac{17}{24},\frac{1}{2}$	$1,\frac{1}{2}$	$\frac{1}{2},\frac{17}{24}$	$\frac{17}{24},\frac{17}{24}$	$\frac{17}{24},\frac{17}{24}$	$1,\frac{17}{24}$	$\frac{1}{2},\frac{17}{24}$	$\frac{17}{24},\frac{17}{24}$	$\frac{17}{24},\frac{17}{24}$	$1,\frac{17}{24}$	$\frac{1}{2},1$	$\frac{17}{24},1$	$\frac{17}{24},1$	1,1
3/4	$\frac{1}{2},\frac{1}{2}$	$\frac{5}{8},\frac{1}{2}$	$\frac{5}{8},\frac{1}{2}$	$1,\frac{1}{2}$	$\frac{1}{2},\frac{5}{8}$	$\frac{5}{8},\frac{5}{8}$	$\frac{5}{8},\frac{5}{8}$	$1,\frac{5}{8}$	$\frac{1}{2},\frac{5}{8}$	$\frac{5}{8},\frac{5}{8}$	$\frac{5}{8},\frac{5}{8}$	$1,\frac{5}{8}$	$\frac{1}{2},1$	$\frac{5}{8},1$	$\frac{5}{8},1$	1,1
4/4	$\frac{1}{2},\frac{1}{2}$	$\frac{1}{2},\frac{1}{2}$	$\frac{1}{2},\frac{1}{2}$	$1,\frac{1}{2}$	$\frac{1}{2},\frac{1}{2}$	$\frac{1}{2},\frac{1}{2}$	$\frac{1}{2},\frac{1}{2}$	$1,\frac{1}{2}$	$\frac{1}{2},\frac{1}{2}$	$\frac{1}{2},\frac{1}{2}$	$\frac{1}{2},\frac{1}{2}$	$1,\frac{1}{2}$	$\frac{1}{2},1$	$\frac{1}{2},1$	$\frac{1}{2},1$	1,1

TABLE II

EXHAUSTIVE ENUMERATION OF ALL 16 POSSIBLE AGENT PAYOFFS, $g^\alpha(\underline{x}=1), g^\alpha(\underline{x}=2)$, FOR BOUNDEDLY INTELLIGENT OPPONENTS HAVING INTELLIGENCE PARAMETER $\alpha = 1/4$, $\alpha = 2/4$, $\alpha = 3/4$, AND $\alpha = 4/4$. SEE TABLE I FOR THE CORRESPONDING f FUNCTIONS AND FOR THE $\alpha = 1$ g FUNCTION. THE PAYOFF FUNCTIONS LABELED IN BOLD ARE THOSE CONSISTENT WITH THE POPULATION $d_2 = \{(1,2;1/2),(2,2;1)\}$.

Y	1/10	2/10	3/10	4/10	5/10	6/10	7/10	8/10	9/10	10/10
$f(\underline{x},\cdot)$			*	*		*	*		*	
d_m^y at $\underline{x}$				*		*			*	
$P(\tilde{g}\mid d_m)$	6/21	5/21	4/21	6/21	0	0	0	0	0	0

Fig. 3. Row 1 indicates the Y values obtainable on a particular payoff function f for each of the $\bar{l}(\underline{x}) = 5$ possible opponent responses. Row 2 gives the Y values actually observed during the training period. Row 3 gives the probabilities of $\tilde{g}$ assuming a uniform probability density across the f which are consistent with d_m. The expected value of $P(\tilde{g}|d_m)$ is 2.48/10.

containing Y values at $\{1/10\} \cup d_m^{y|\underline{x}}$.[14] Since the total number of possibilities consistent with the data is $\binom{|Y|-k}{\bar{l}(\underline{x})-k}$ this fraction is $\binom{|Y|-k-1}{\bar{l}(\underline{x})-k-1}/\binom{|Y|-k}{\bar{l}(\underline{x})-k} = (\bar{l}(\underline{x})-k)/(|Y|-k)$. Similarly, $P(\tilde{g} = 2/10|d_m)$ is $\binom{|Y|-k-2}{\bar{l}(\underline{x})-k-1}/\binom{|Y|-k}{\bar{l}(\underline{x})-k}$ because we know that the function can not contain a sample having fitness less than 2/10.

Thus, in the general case, we have

$$\pi_{k,r}(\tilde{g}) = \frac{1}{\binom{a}{b}}\left\{\theta(r-\tilde{g})\binom{a-|Y|\tilde{g}}{b-1} + \delta_{\tilde{g},r}\binom{a-|Y|r+1}{b}\right\}$$

where $a = |Y| - k$, $b = \bar{l}(\underline{x}) - k$, $\theta(x) = 1$ iff $x > 0$, and $\delta_{\tilde{g},r} = 1$ iff $\tilde{g} = r$. Since it is easily verified that

$$\binom{a-|Y|r+1}{b} + \sum_{\tilde{g}'=1}^{|Y|r-1}\binom{a-\tilde{g}'}{b-1} = \binom{a}{b}$$

this probability is normalized correctly. The expected value of $\tilde{g}$ is therefore

$$\mathbb{E}(\tilde{g}|d_m) = \frac{1}{|Y|\binom{a}{b}}\left\{r\binom{a-|Y|r+1}{b} + \sum_{\tilde{g}'=1}^{|Y|r-1}\tilde{g}'\binom{a-\tilde{g}'}{b-1}\right\}.$$

Evaluating this sum we find

$$\begin{aligned}\mathbb{E}(\tilde{g}|d_m) &= \left[|Y|\binom{a}{b}\right]^{-1}\left\{\binom{a+1}{b+1} - \binom{a+1-|Y|r}{b+1}\right\}\\ &= \frac{|Y|^{-1}}{b+1}\,\frac{(a+1)^{\underline{b+1}} - (a+1-|Y|r)^{\underline{b+1}}}{a^{\underline{b}}}\end{aligned}$$

where the falling power, $a^{\underline{b}}$, is defined by $a^{\underline{b}} \equiv a(a-1)(a-2)\cdots(a-b+1)$. For the case at hand where $|Y| = 10$, $\bar{l}(\underline{x}) = 5$, and $k = 3$ we have $a = 7$ and $b = 2$. Since $r = 4/10$ the expected value is $\mathbb{E}(\tilde{g}|d_m) = \frac{1}{10}(8^{\underline{3}} - 4^{\underline{3}})/(3 \cdot 7^{\underline{2}}) = 52/21 \approx 2.48/10$.

[14]By $d_m^{y|\underline{x}}$ we mean the set of Y values sampled at $\underline{x}$.

C. *Determination of $\pi_{k,r}(\tilde{g})$: non-distinct Y*

In Figure 4 we present another example where $\bar{l}(\underline{x}) = 5$, $k = 3$, and $r = 4/10$. In this case, however, there are duplicate Y values. The total number of functions consistent with the data is $|Y|^{\bar{l}(\underline{x})-k} = |Y|^b$. In this case it is easiest to begin the analysis with the case $\tilde{g} = r$. The number of functions having the minimum of the remaining b points equal to $|Y|$ is 1. Similarly, the number of functions having a minimum value of $(|Y|-1)$ is $2^b - 1$. 2^b counts the number of functions where the b function values can assume one of Y or $Y-1$. The -1 accounts for the fact that 1 of these functions has a minimum value of Y and not $Y-1$. Generally, the number of functions having a minimum value of r' is $(|Y| - |Y|r' + 1)^b - (|Y| - |Y|r')^b$. All $r' \geq r$ will result in the minimal observed value r so that the total number of functions having an observed minimum of r is

$$\sum_{r'=r}^{|Y|}\left[(|Y| - |Y|r' + 1)^b - (|Y| - |Y|r')^b\right] = (|Y| - |Y|r + 1)^b.$$

Thus the probability of $\tilde{g} = r$ is

$$\pi_{k,r}(\tilde{g} = r) = |Y|^{-b}(|Y| - |Y|r + 1)^b.$$

We turn now to determining the probabilities where $\tilde{g} < r$.

Of the b remaining Y values the probability that the minimum is $\tilde{g}$ is

$$\pi_{k,r}(\tilde{g}) = |Y|^{-b}\left\{(|Y| - |Y|\tilde{g} + 1)^b - (|Y| - |Y|\tilde{g})^b\right\}.$$

Combining these results we obtain the final result

$$\pi_{k,r}(\tilde{g}) = \theta(r-\tilde{g})\left\{\left(1 - \tilde{g} + \frac{1}{|Y|}\right)^b - (1-\tilde{g})^b\right\} + \delta_{r,\tilde{g}}\left[1 - r + \frac{1}{|Y|}\right]^b.$$

Y	1/10	2/10	3/10	4/10	5/10	6/10	7/10	8/10	9/10	10/10
$f(\underline{x},\cdot)$			*	*		**			*	
d_m^y at $\underline{x}$				*		**				
$P(\tilde{g}\|d_m)$	19/100	17/100	15/100	49/100	0	0	0	0	0	0

Fig. 4. Row 1 indicates the Y values obtainable on a particular payoff function f for each of the $\bar{l}(\underline{x}) = 5$ possible opponent responses. Row 2 gives the Y values actually observed during the training period. Row 3 gives the probabilities of $\tilde{g}$ assuming a uniform probability density across the f which are consistent with d_m. Note that unlike Fig. 3 there are some duplicate Y values. The expected value of $P(\tilde{g}|d_m)$ is 2.94/10.

Given $\pi_{k,r}(\tilde{g})$ the expectation value of $\tilde{g}$ is found as

$$\mathbb{E}(\tilde{g}|d_m) = r\Big(1 - r + \frac{1}{|Y|}\Big)^b + \sum_{\tilde{g}=1/|Y|}^{r-1/|Y|} \tilde{g}\Big\{\Big(1-\tilde{g}+\frac{1}{|Y|}\Big)^b - (1-\tilde{g})^b\Big\}$$
$$= \sum_{r'=1/|Y|}^{r} \Big(1 - \Big(r' - \frac{1}{|Y|}\Big)\Big)^b$$

where we have cancelled appropriate terms in the telescoping sum. If we define $S_k(n) \equiv \sum_{i=1}^n i^k$ then we can evaluate the last sum to find

$$\mathbb{E}(\tilde{g}|d_m) = |Y|^{-b}\{S_b(|Y|) - S_b(|Y| - |Y|r)\}.$$

Though there is no closed form expression for $S_k(n)$, a recursive expansion of $S_k(n)$ in terms of $S_j(n)$ for $j < k$ is

$$S_k(n) = \frac{1}{k+1}\Big\{n^{k+1} - \sum_{j=0}^{k-1}(-1)^{k-j}\binom{k+1}{j}S_j(n)\Big\}.$$

The recursion is based upon $S_0(n) = n$.

In the concrete case above where $|Y| = 10$, $r = 4/10$, and $b = 2$ the expected value is $\frac{1}{10}294/100 = 2.94/10$.

D. Continuum Limit

In the limit where $|Y| \to \infty$ we can approximate the expectation $\mathbb{E}(\tilde{g}|d_m)$ given by the sum

$$\mathbb{E}(\tilde{g}|d_m) = \sum_{r'=1/|Y|}^{r} (1 - (r' - 1/|Y|))^b = \sum_{r'=0}^{r-1/|Y|} (1-r')^b$$

by the integral

$$\mathbb{E}(\tilde{g}|d_m) = \int_0^r dr'\ (1-r')^b = \frac{1}{b+1}\{1 - (1-r)^{b+1}\}. \tag{6}$$

The prediction made by this approximation at $|Y| = 10$, $r = 4$, and $b = 2$ is $2.61/10$ as opposed to the correct result of $2.94/10$. However, had $|Y| = 1000$ and $r = 400$ the accurate result would have been 261.65/1000 while the approximation gives 261.3/1000.