Coevolutionary Free Lunches

David H. Wolpert and William G. Macready

Abstract—**Recent work on the foundational underpinnings of black-box optimization has begun to uncover a rich mathematical structure. In particular, it is now known that an inner product between the optimization algorithm and the distribution of optimization problems likely to be encountered fixes the distribution over likely performances in running that algorithm. One ramification of this is the "No Free Lunch" (NFL) theorems, which 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 most optimization scenarios. In addition to the optimization scenarios addressed in the NFL results, this framework covers multiarmed bandit problems and evolution of multiple coevolving 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 multiplayer 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. However, in the typical coevolutionary scenarios encountered in biology, where there is no champion, the NFL theorems still hold.**

Index Terms—**Coevolution, multiarmed bandits, no free lunch, optimizations, self-play.**

I. Introduction

RECENT work has started to uncover the mathematical foundations of black-box optimization, i.e., the foundations of optimization scenarios where one does not know the "objective/fitness" function to be optimized, but can only work with samples of that function [5], [9], [10], [14], [17], [21]–[23], [28], [31]. In particular, it is now known that an inner product between one's black-box optimization algorithm and the distribution of likely objective functions fixes the distribution over likely performances in running that algorithm. An immediate implication of this result is the "No Free Lunch" (NFL) theorems. These establish 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 [26]. There has also been much work extending these early results to different types of optimization (e.g., to multiobjective optimization [7]). 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 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 [13].

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 multiplayer game. The goal is to train that champion player to perform as well as possible in that subsequent game. (For example, in checkers and chess, see [4] and [15].)

Early work on coevolutionary scenarios includes [2], [16], and [29]. 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 [19]). Coevolution in these cases sometimes 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, *generalized optimization* (GO), that covers both traditional optimization and coevolutionary scenarios. GO also covers other scenarios such as multiarmed bandits. We then use GO to explore the differences between traditional optimization and coevolution. We find dramatic differences between the traditional optimization and coevolutoin scenarios. In particular, unlike the fundamental NFL result for traditional optimization, in the self-play version of coevolution 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, the NFL theorems still hold, i.e., uniformly averaged over all objective functions, all algorithms perform identically.

Section II summarizes the previous NFL work that we extend, and Section III motivates these extensions. Section IV presents the resultant extended NFL framework, GO, and provides example illustrations of GO. Section V applies GO to self-play, and Section VI demonstrates that NFL results need not apply in this case. In Section VII, we discuss the role of the antagonist's intelligence in such free lunch results. We conclude in Section VIII.

Manuscript received August 11, 2004; revised January 15, 2005.

D. H. Wolpert is with the NASA Ames Research Center, Moffett Field, CA 94035 USA (e-mail: dhw@email.arc.nasa.gov).

W. G. Macready is with D-Wave Systems, Vancouver, BC V6J 4Y3, Canada (e-mail: wgm@dwavesys.com).

Digital Object Identifier 10.1109/TEVC.2005.856205

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

II. Background

Motivated by the myriad heuristic approaches to black-box combinatorial optimization, researchers have sought insight into how best to match optimization algorithms to problems. The importance of this task was highlighted in [36]. We review the approach taken in that paper as it forms the starting point for our 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 GO, the generalized version of the original NFL framework.) However, for pedagogical simplicity here, we restrict attention to single-valued f's. The goal of one's "black-box optimization/search algorithm" is to find an x that has a good value of $f(x)$, using only samples of f. In particular, we are interested in the performance of such algorithms when averaged across some distribution $P(f)$ of single-valued fitness functions.

The mathematical formalization of what we mean by "black-box optimization/search algorithm" in [36] is motivated by iterative algorithms like evolutionary 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": $d_m \equiv \{(d_m^x(1), d_m^y(1)), \ldots, (d_m^x(m), d_m^y(m))\}$. We refer to this ordered set of configurations as a "sample".[2] Configurations in d_m are ordered according to the time at which the algorithm sampled them. Thus, $d_m^x(t)$ is the tth sampled x value, and $d_m^y(t) = f(d_m^x(t))$ is the associated fitness value. The ordered sets of all X and Y values are 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 evolutionary 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. The fact that we are comparing them that way means that *for the purposes of our analysis*, we can treat them as though each point they visit is distinct. Further discussion of this point is found in [36].

Based upon these definitions an algorithm is a (perhaps nondeterministic) 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 $x \in d_m^x$.[3] The algorithm begins with the selection of a starting configuration as specified by an initial distribution $P_1(d_1^x(1) = x)$.

Combining these considerations, an algorithm a is a specification of the probability distributions P_1, P_2, etc. (This definition of a search algorithm has been reused by others in the special case where the mapping was assumed to be deterministic, e.g., in [30].) 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, evolutionary algorithms, tabu search, etc.), not all algorithms are of this type. In particular, algorithms that use more information about the objective function than just the samples are not 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. Examples of such a measure are the best fitness value seen so far, the ending fitness value, etc. (It is important to note that use of such a 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,[4] 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 [28], [30], [36]. Thus, the expected performance of any pair of algorithms is equal under that type of average. This is an example of an NFL result. The most general form for $P(f)$ for which such NFL results hold is derived in [11], [21].

Reference [36] considers many extensions of the basic NFL results. It is shown there that algorithms may be distinguished once we look beyond simple average performances. In addition, one can derive results independent of the distribution over optimization functions $P(f)$ [23].

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, the extension we arrive at, GO, addresses many problems of interest in both evolutionary and coevolutionary optimization. Before presenting the details of GO, we motivate its extensions through consideration of an idealized self-play optimization problem, and the k-armed bandit.

[2] This set was called a trace in [30].

[3] For algorithms that in actuality revisit points, P_m is the distribution of what new point they product when they finally are done revisiting old points.

[4] So, in particular, $P(f)$ for any permutation $\sigma : X \to X$ of inputs $P(f) = P(f \circ \sigma)$.

III. Motivation

In this section, we present two separate motivations for our generalization of the original NFL framework into GO.

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 $\bar{x}$.

To illustrate this consider a multistage game involving the two players [1], [18], such as checkers [4]. Have the players be computer programs. In this setting, the moves $\bar{x}$ 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.[5] In this case, f reflects the performance of one particular one of the players. As another example, $\underline{x}$ might represent an algorithm to sort a list, and $\bar{x}$ a mutable set of lists to be sorted. The payoff f here reflects the ability of the algorithm to sort the lists in $\bar{x}$.

In self-play, we fix attention on the payoff to one of the two players, the "champion," with the other player being the "opponent." A fitness function $f(\underline{x}, \bar{x})$ gives the reward to the champion (e.g., $+1$ for a pair of moves in which it wins, 0 for an indeterminate or drawing pair, and -1 for a losing pair). Now, concatenate the moves of our player $(\underline{x})$ and the opponent $(\bar{x})$ into a single joint point $x = [\underline{x}, \bar{x}]$. The mathematical advantage of doing this is that it means we will not need to generalize the definition of fitness functions when we extend the NFL framework into GO; the fitness function remains a mapping from $X = \underline{X} \times \bar{X}$ into Y. Now, however, X is the space of joint (champion, opponent) game moves.

Generalizing from the two-player version of self-play, we can have several players compete in a tournament, and from the results of that tournament, we select a single best player. That best player 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 tournament designs, and to see if NFL-like results hold for such designs.

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' moves 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's move.

We use nomenclature analogous to the original NFL work to indicate how training games are selected. The m distinct training games and their fitnesses are indicated as $d_m \equiv \{(d_m^x(1), d_m^y(1)), \ldots, (d_m^x(m), d_m^y(m))\}$. Similarly, we parallel the original NFL work and write the probabilistic mapping that selects each new training game's moves based on the results of the preceding ones as a set of conditional distributions. We write that set of distributions as the "algorithm" a, exactly as in the original NFL framework.

The second component of designing a tournament is choosing the champion's move at the end of the tournament. We indicate how that is done with a function A, which maps a completed sequence of training games d_m into the champion's move. We parameterize that champion's move as the associated subset of all joint moves x consistent with it. For example, say that after our set of games our champion will take the role of the first player in a two-player subsequent game with a single antagonist. In other words, our champion is a choice of a (hopefully) optimal first player's move for that subsequent game. So, we choose that champion's move by selecting a particular value $\underline{x}^*$ for the move $\underline{x}$ of the first player in the subsequent game. Since that choice of move doesn't restrict the antagonist's responses, we indicate that move as the subset of all $x \in X$ with $\underline{x} = \underline{x}^*$, i.e., the subset $\{(\underline{x}^*, \bar{x}) | \bar{x} \in \bar{X}\}$. So, A maps d_m to such a subset of X. (In the more general approach of Section V, A is allowed to map d_m to probability distributions over X, not just to 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 that champion. If the champion plays the game according to $\underline{x}^*$, then this worst case measure may be written as $\min_{\bar{x}} f(\underline{x}^*, \bar{x})$ where $\bar{x}$ ranges over all possible opponent moves. Our definition of $A(d_m)$ allows us to write that worst case performance more generally 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_m^y (the observed game outcomes), independent of f. 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.

Of course, there are other ways of quantifying the performance of the champion besides the maximin value, and in some cases those alternatives are more appropriate. 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. Subtleties in evaluating the performance of game-playing moves are considered in [3], [8], and [12].

B. Bandit Problems

The k-armed bandit problem is simple, but captures much of the essence of the critical exploration/exploitation tradeoff inherent in black-box optimization. In this problem a player is faced with choosing repeatedly between k distinct real-valued stochastic processes having different means. With each selection it makes (of either process 1, process 2, …, process k)

[5] In noncooperative game theory these programs are called "normal form strategies." To minimize terminology, here we will refer to them as moves always, even when (as in evolutionary game theory) this is uncoventional.

the player receives a real-valued reward sampled stochastically from the process it chose. The player'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 [20], [24].

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 of "fitness function" from a $X \rightarrow Y$ mapping to a mapping from $X \rightarrow Z$, where Z is a space capturing probabilistic models. This is illustrated below.

IV. Generalized Optimization Framework

As seen in the preceding pair of motivational examples, to increase the scope of NFL-like analyses, we need to make two slight extensions. First, we must broaden the definition of performance measures to allow for dependence on f, and second, we need to generalize fitness functions to allow for nondeterminism. The resultant framework, GO, is closely related to the one used in the very first work on NFL, preceding the NFL results for the problem of search. This original work was the application of NFL to supervised machine learning [32]–[34].[6]

A. Formal Specification of Generalization Optimization

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

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; different x's result in different distributions over 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 generalization of such single-valued mappings by introducing Z into GO is what allows for noisy payoffs, and to allow GO to cover bandit problems.

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

$$d_m \equiv (d_m^x, d_m^z) \equiv \left(\{d_m^x(t)\}_{t=1}^m, \{d_m^z(t)\}_{t=1}^m\right).$$

As in classic NFL, each $d_m^x(t)$ is a particular $x \in X$. Each $d_m^z(t)$ is a (perhaps stochastic) function of $f(d_m^x(t)), F(f(d_m^x(t)))$. For example, say z's—values of $f(x)$—are probability distributions over reward values. Then, $d_m^z(t) = F(f(d_m^x(t)))$ could consist of the full distribution $f(d^x(t))$, i.e., $F(\cdot)$ could be the identity mapping. Alternatively, $d_m^z(t)$ could consist of a moment of the distribution $f(d_m^x(t))$, or even a random sample of it. In general, we allow the function F specifying $d_m^z(t)$ to vary with t. However, that freedom will not be exploited here. Accordingly, to minimize the notation, we will leave the space that each $d_m^z(t)$ lives in implicit, and will also leave the function F implicit; the symbol "F" will not recur below. As shorthand, we will write $d(t)$ to mean the pair $(d_m^x(t), d_m^z(t))$.

A *search algorithm*, a (or just "algorithm" for short), is an initial distribution $P_1(d_m^x(1))$ of the initially selected point $d_m^x(1) \in X$, together with a set of $m-1$ separate conditional distributions $P_t(d_m^x(t) \mid d_{t-1})$ 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. Often, as in previous NFL work, we assume that the next x has not been previously seen. This is reflected as an implicit restriction on the conditional distributions $P_t(d_m^x(t) \mid d_{t-1})$.

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 $d_m^z(t)$'s from the $f(d_m^x(t))$'s. This expectation $\mathbb{E}(C)$ is formed by averaging over any stochasticity in the mapping from f's to associated $d_m^z(t)$'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 in a GO scenario—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. (We direct the reader to the original proofs of the NFL theorems; the extensions of those proofs to the GO setting is straightforward.) 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 GO 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). This allows for the treatment of noisy fitness functions, where the fitness at any x value is sampled from an x-dependent probability distribution.

As remarked above, deterministic fitness functions are the special case where such distributions must be delta functions. In this case the implicit mapping from $f(d_m^x(t))$ to the associated $d_m^z(t)$ is given simply by evaluating the real value f has at $d_m^x(t)$. As an alternative formulation, for such deterministic 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 this case, $d_m^z(t)$ is simply the value $f(d_m^x(t))$. In our more general version of the original NFL scenario, z is a nondelta function

[6] The NFL theorems for optimization are sometimes seen as having implications for natural selection. In contrast, the philosophical domain most strongly impacted by the original NFL theorems for supervised learning is the problem of justifying inductive inference of scientific theories from data, e.g., as considered by Hume.

distribution over Y, and the mapping $f(d^x_m(t))$ to the associated $d^z_m(t)$ is given by forming a stochastic sample of $f(d^x_m(t))$.

In the scenario of the original NFL research, 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 theorems these conditions are crucial: The cost function must not depend on f, and the search algorithm a must not allow revisit. Both conditions apply to the version of GO given here.

More generally, the NFL theorems apply to GO scenarios which can be cast as an instance of this example. This fact will be exploited below.

Example 2: The formal specification of two-player self-play in terms of GO 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. However, the variables are interpreted differently, e.g., x is now a joint move.

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's move to be used in the subsequent game against the antagonist. As we have seen, the procedure for making this choice is a function $A(d_m)$ mapping d_m to a subset of X. Since it measures performance of the champion's move against the antagonist, C must involve this specification of the champion's move.

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

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

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

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

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, this definition of cost still applies if the number of players in any game is greater than two (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 except in this specification of C. Finally, note that the C of self-play depends on f.

Say we have a two-player self-play scenario and the antagonist has no care for any goal other than hurting our champion. Say that the antagonist is also omniscient (or at least very lucky), and chooses the $\bar{x}$ which achieves its goal. Then, the expected reward to the champion is given by (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, and all associated d_m generated by running our algorithm. 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 [20], and analyzed thoroughly in [24]. The scenario for that problem is identical to that of Example 1, 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 [6], [25], [35], there is a set of "players" who change their moves from one game to the next, just like in self-play. Unlike in the self-play scenario though, in the general biological coevolution scenario, 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 in GO. Moreover, the formulation for the general coevolution scenario involves definitions of Z, f, etc., that would appear counterintuitive if we were to interpret them the same way we do in self-play.

We formulate the general coevolution scenario in GO by having a set of N agents (or agent types, or player types, or cultures, or lineages of genomes, or lineages of genes, etc.), just like in self-play. Their move spaces are written S_i. Unlike in self-play (where there are also multiple agents' move spaces), X is extended beyond the current joint move to include the joint "population frequency" value of those moves. Formally, we write

$$X = (S_1, u_1) \times \cdots \times (S_N, u_N) \tag{4}$$

and interpret each $s_i \in S_i$ as a move of agent type i and each $u_i \in \mathbb{R}$ as a frequency with which agent i occurs in the overall population of all players. Implicitly, all instances of agent i in the population play the same move s_i.[7] As an example, we could have each i be a single agent, so that "learning" by agent i corresponds to having the move of $i, s_i \in S_i$, change from one time step to the next. In this example, the population frequencies are confined to the two values $\{0, 1/N\}$. (For example, if we have no death or birth processes, then the frequencies are always $1/N$.) As another example, when i is a "lineage of a gene," it is not the move s_i (i.e., the gene) that changes from one time step to the next, but the associated frequency of that move in the population. This too is accommodated 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-move. More generally, our formulation allows both kinds of changes to occur simultaneously. In addition, mutation, e.g., modification of the gene, can be captured in this GO formulation. This is done by having some i's that at certain times have zero population frequency, but then stochastically jump to nonzero frequency, representing a new agent that is a mutant of an old one.

To be more precise, for any t, we interpret $s_i(t)$ as i's current move. However, we interpret $u_i(t)$ as i's *previous* population frequency, i.e., the population frequency, at the preceding time

[7] 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.

step, of the move that i followed then. In other words, we interpret the u_i component of $d_m^x(t)$ as the population frequency at time step $t-1$ of the move followed by agent i at $t-1$, a move given by the S_i component of $d_m^x(t-1)$. So the information concerning each agent i is "staggered" across pairs of successive time steps. This is done as a formal trick, so that a can give the sequence of joint population frequencies that accompanies the sequence of joint moves, as described below.

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 move 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 move 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 move $s_1 \times \ldots s_N$ and the values of all agents' immediately preceding population frequencies, and based on that determines the new value of each agent's population frequency. More precisely, to accommodate finite-population size effects, $f(d_m^x(t))$ is that distribution over possible current population joint-frequencies, and $d_m^z(t)$ is a sample of that distribution.

In this general coevolution scenario, our choice for a, which produces $d_m^x(t+1)$ from $d_m(t)$, plays two roles. These correspond to its updates of the move components of $d_m^x(t)$ and of the population frequency components of $d_m^x(t)$, respectively. More precisely, one of the things a does is update the population frequencies from those of the previous time-step $t-1$ (which are stored in $d_m^x(t)$) to the ones given by $d_m^z(t)$. This means a directly incorporates those population frequencies into the $\{u_i\}$ components of $d_m^x(t+1)$.

The other thing a does, as before, is determine the joint move $[x_1, \ldots, x_N]$ for time $t+1$. At the risk of abusing notation, as in self-play, we can write the generation of the new move 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 move 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.[8] However, in the simplest version of evolutionary game theory, the joint move 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.

As always, the choice of C depends on the research problem at hand. It is set by 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, one for each agent, each such component depending only on the associated agent's components of d_t. As an example, often in population biology each component is j's population frequency at $t-1$; what one wishes to research is the population frequencies at the end of a sequence of interactions between the biological entities.

Usually, in such biological research, there is no notion of a champion being produced by the sequence of interactions and subsequently pitted against an external antagonist in a "bake-off." (Famously, evolution is not teleological.) Accordingly, unlike in self-play, in such research there is no particular significance to results for alternative choices of C that depend on f. So make the approximation, reasonable in real biological systems, that x's are never revisited. Then, our coevolutionary scenario is a special case of Example 1. (More precisely, it is a special case of the choices of Z, f, etc., of Example 1.) This means that the NFL theorems hold for such choices of C, under our never-revisiting approximation.

Some authors have promoted the use of the general coevolution scenario as a means of designing an entity to perform well, rather than as an analytic tool for analyzing how a biological system happens to develop. For example, this is the premise underlying much of the use of genetic algorithms for optimization. In general, whether or not NFL applies to such a use of coevolution for design will depend on the details of the design problem at hand.

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 well as an optical sensor. Then, even if we are in the general coevolutionary scenario of interacting populations, we can still cast the problem as a special case of Example 1. In particular, for our design problem C does not involve any "subsequent game against an antagonist," so C is independent of f. (Just like in Example 1, C only depends on f indirectly, through the characteristics of the population produced for f. Unlike in self-play, there is no direct dependence on f in C.) Similarly, we can restrict search to never be revisiting.

Due to the fact that they're a special case of Example 1, the NFL theorems hold in such scenarios. The extra details of the dynamics introduced by the general biological coevolutionary process do not 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 nonzero population frequency) in the years after a major change to the ecosystem. More precisely, say that our problem is to design that organism, and then, *after we are done* its ecosystem is subjected to that change, a change we know nothing about *a priori*. 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." Now, it may be quite appropriate to choose a C that depends directly on f. In this situation NFL may not hold.

[8] Obvious elaborations of GO 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.

There are other ways one can express the general coevolution scenario in GO, i.e., other choices for the roles of f, a, etc. that capture that scenario. The advantage of the one described here is how it formally separates the different aspects of the problem. f plays the role of the laws of Nature which map joint moves and population frequencies to new population frequencies (e.g., the replicator dynamics). All variability in how one might update moves—crossover, 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 moves) are paired against each other in a (perhaps stochastically formed) sequence to generate a set of two-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 move $\underline{x}$ and it's opponent plays $\bar{x}$ is written as $f(x)$, where $x = (\underline{x}, \bar{x})$ is the joint move. We make no assumption about the structure of moves except that they are finite.

We define the payoff for the agent playing move $\underline{x}$ independent of an opponent's reply, $g(\underline{x})$, as the least payoff over all possible opponent responses: $g(\underline{x}) \equiv \min_{\bar{x}} f(\underline{x}, \bar{x})$. With this criterion, the best move an agent can play is that move 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 moves of the agent and its opponents. (Note that whether that opponent varies or not is irrelevant, since we are setting its moves.) 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 move in the competition is better than the worst possible move. The only way to ensure an outcome guaranteed to be better than the worst possible is to exhaustively explore all possible responses to move $\underline{x}$, and then determine that the worst value of f for all such joint moves is better than the worst value for some other move $\underline{x}'$. To do this requires that m is greater than the total number of possible moves available to the opponent, but even for very large m, unless all possible opponent responses have been explored, we cannot 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 move $\underline{x}$ is $g(\underline{x})$ and that another joint move $x' = (\underline{x}', \bar{x}')$ 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 moves, 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 $\underline{l}$ moves available to an agent, and label these using $\underline{X} \equiv [1, \ldots, \underline{l}]$. For each such move, we assume the opponent may choose from one of $\bar{l}(\underline{x})$ possible moves forming the space $\bar{X}(\underline{x})$.[9] Consequently, the size of the joint move space is $|X| = \sum_{\underline{x}=1}^{\underline{l}} \bar{l}(\underline{x})$. For simplicity, we take $\bar{X}(\underline{x})$ to be independent of $\underline{x}$ so that $\bar{X} = [1, \ldots, \bar{l}]$ and $|X| = \underline{l}\bar{l}$. If the training period consists of m distinct joint moves, even with m as large as $|X| - \underline{l}$, we cannot guarantee that the agent will not choose the worst possible move in the competition as the worst possible move could be the opponent response that was left unexplored for each of the $\underline{l}$ possible moves.

As always, a sample of configurations (here configurations are joint moves) 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 moves. 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 $f : X \rightarrow Y$, where $X = \underline{X} \times \bar{X}$ is the space of joint moves. As in the general framework, a sample of size m is represented as

$$d_m = \{(d_m^x(1), d_m^y(1)), \ldots, (d_m^x(m), d_m^y(m))\}$$

where $d_m^x(t) = \{d_m^{\underline{x}}(t), d_m^{\bar{x}}(t)\}$, and $d_m^y(t) = f(d_m^{\underline{x}}(t), d_m^{\bar{x}}(t))$ and $t \in [1, \ldots, m]$ labels the samples taken. In the above definition, $d_m^{\underline{x}}(t)$ is the tth move adopted by the agent, $d_m^{\bar{x}}(t)$ 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 [36], 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 move to apply in the competition based upon the results of the training samples. In GO, this choice is buried in the performance measure through the function $A(d_m)$. Recall that $A(d_m)$ is a

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

TABLE I

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

$(\underline{x}, \bar{x})$	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

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 move $\underline{x}^*$ based on d_m, then the subset output by A is $\{(\underline{x}^*, \bar{x}) \mid \bar{x} \in \text{possible opponent responses to } \underline{x}^*\}$.

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 nondeterministic A. In such cases, $A(d_m)$ might stochastically define an optimal move $\underline{x}^*$ 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

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

where the $\min$ over $\bar{x}$ 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, $\sum_f P(C \mid f, m, a)$ is independent of a.[10] A natural extension of this result considers a nonuniform average over fitness functions. In this case, the quantity of interest is $\sum_f P(C \mid f, m, a) P(f)$, where $P(f)$ weights different fitness functions.

A result akin to this one in the self-play setting would state that the uniform 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 moves 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 moves, i.e., $\underline{X} = \{1, 2\}$, the opponent has two responses for each of these moves, i.e., $\bar{X} = \{1, 2\}$, 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 $g = [1/2 \ 1/2]$, $3/16$ have $g = [1/2 \ 1]$, $3/16$ have $g = [1 \ 1/2]$, and $1/16$ have $g = [1 \ 1]$, where $g = [g(\underline{x} = 1) \ g(\underline{x} = 2)]$.

If we consider a particular sample, say $d_2 = \{(1,2;1/2),(2,2;1)\}$, the payoff functions that are consistent with this sample 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 sample 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.

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 $1/|Y|, 2/|Y|, \ldots, 1$. 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 moves offer the same number of possible opponent responses $\bar{l}$. We consider algorithms that explore $m = \underline{l}$ distinct joint samples. Agent moves are labeled by $\underline{x} \in \{1, \ldots, \underline{l}\}$ and opponent responses are labeled by $\bar{x} \in \{1, \ldots, \bar{l}\}$. For simplicity, we take $\underline{l} = \bar{l}$.

In the following section, we consider three different algorithms and show different expected performance for each 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.

[10] Actually, far more can be said, and the reader is encouraged to consult [36] for details.

A. Algorithms Having Different Expected Performance

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

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 $|Y|^{\underline{l}\bar{l}}$ functions.[11] We note that for any given agent move $\underline{x}$, the $|Y|^{\underline{l}\bar{l}}$ possible function values at the joint moves $(\underline{x},\cdot)$ are replicated $|Y|^{\underline{l}\bar{l}}/|Y|^{\bar{l}} = |Y|^{\bar{l}(\underline{l}-1)}$ times. The number of times that a $g(\underline{x})$ value of $1-i/|Y|$ is attained in the first $|Y|^{\bar{l}}$ distinct values is $(i+1)^{\bar{l}}-i^{\bar{l}}$. Thus, the average $g(\underline{x})$ value, which we denote $\langle g\rangle$, is

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

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

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

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 $m=\bar{l}$, there are $|Y|^{\bar{l}}$ 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 moves $\underline{x}\neq 1$. This average is taken over the $|Y|^{\bar{l}(\underline{l}-1)}$ 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 moves $\underline{x}\neq 1$). 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

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

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

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

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

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

In the continuum limit, we have

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

where we have recalled the expected value $\langle g\rangle = 1/(1+\bar{l})$. We note that as $\bar{l}\to\infty$ 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 move for the agent is to choose that $\underline{x}^*$ which has the largest fitness observed in the sample. With this insight, we observe that when summing over all functions, there are $|Y|^{\bar{l}-1}$ possible completions to $\max d_m^y$ 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

$$\langle g\rangle_2 = \frac{1}{|Y|^{\bar{l}}}\sum_{d_m^y}\sum_{i=0}^{|Y|-1}\min\left(\max d_m^y, 1-\frac{i}{|Y|}\right)n_{\bar{l}-1}(i).$$

We proceed in the same fashion as above by defining[12] $i_d \equiv |Y|(1-\max d_m^y)$ (which depends on d_m^y) so that

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

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

$$\begin{aligned}\langle g\rangle_2 = \sum_{j=1}^{|Y|}\Bigg[&\frac{j}{|Y|}\left\{1-\frac{j}{|Y|}\right\}^{\bar{l}-1}\\ &+\sum_{i=|Y|-j}^{|Y|-1}\frac{|Y|-i}{|Y|}n_{\bar{l}-1}(i)\Bigg]\,n_{\bar{l}}(j-1)\end{aligned}$$

[11] Recall that $|X| = \underline{l}\bar{l}$.

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

The continuum limit in this case is found as

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

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

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

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

Fig. 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 $|Y|^{-\underline{l}\bar{l}}\sum_f \theta(\mathrm{perf}_1(f) - \mathrm{perf}_2(f))$, where $\mathrm{perf}_1(f)$ is the performance of algorithm a_1 on payoff function f, $\mathrm{perf}_2(f)$ 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[13]

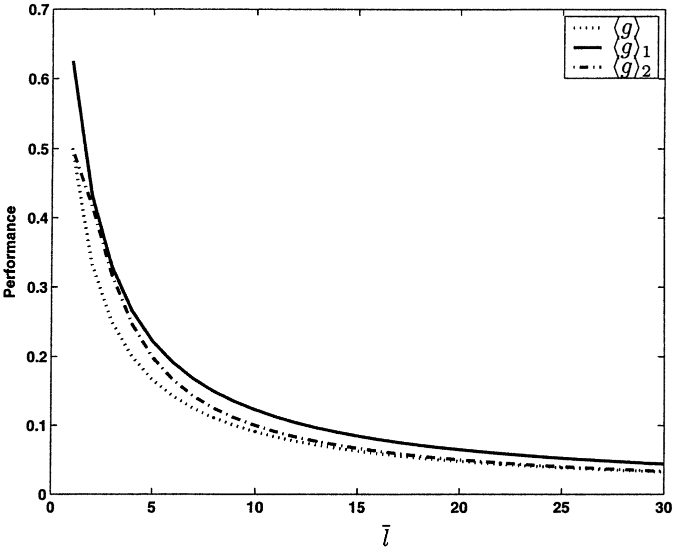

Fig. 1. Expected performance of algorithm a_1 (indicated as $\langle g\rangle_1$), which exhaustively enumerates the opponents response to a particular move, and algorithm a_2 (indicated as $\langle g\rangle_2$), which samples only one opponent response to each move. For comparison, we also plot $\langle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

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

Similarly, the performance of algorithm a_2 is given by

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

where $\underline{x}_2^*$ is the move 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:

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

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

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

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

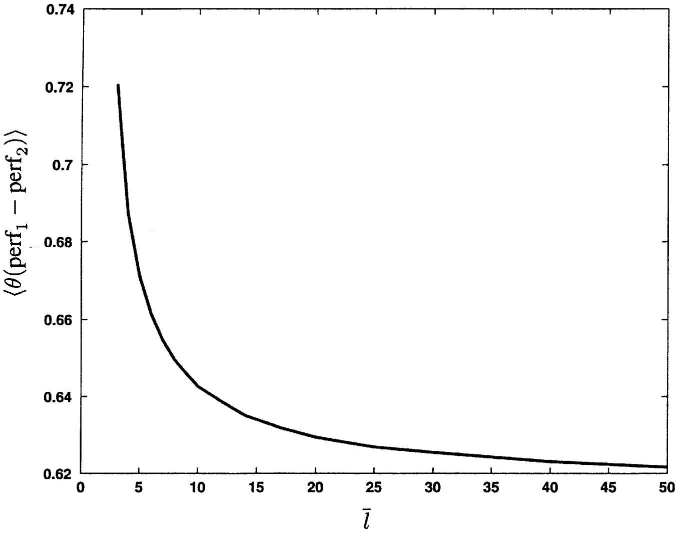

Fig. 2. The fraction of functions in the continuum limit where algorithm a_1, which exhaustively enumerates the opponents response to a particular move, performs no worse than algorithm a_2, which samples only one opponent response to each move. The figure was generated with 10^7 Monte Carlo samples of the integrand for each value of $\bar{l}$.

and

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

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 Fig. 2, which plots the fraction of functions for which $\mathrm{perf}_1 \geq \mathrm{perf}_2$. 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 move with the highest expected $g(\underline{x})$), and showed different performance. Unsurprisingly, we can construct algorithms with different expected performance that 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 $\underline{A}$.[14] 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 move that maximizes this expected value while the worst champion-selection rule selects the move 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 move $\underline{x}$ 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 sample 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.[15] Since we are concerned with the worst possible 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., $r(\underline{x}, d_m) = \min_{\bar{x} \in d_m^{\bar{x}}} d_m^y(\underline{x}, \bar{x})$. 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, 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)

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

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

The best move given the training sample is the deterministic choice $\underline{A}_{\text{best}}(d_m) = \arg\max_{\underline{x}} C_m(\underline{x})$ and the worst move is $\underline{A}_{\text{worst}}(d_m) = \arg\min_{\underline{x}} C_m(\underline{x})$. In the example of Section V-B with the sample 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,

[14] 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 $A(d_m)$.

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

TABLE II

EXHAUSTIVE ENUMERATION OF ALL 16 POSSIBLE AGENT PAYOFFS, $g^\alpha(\underline{x}=1), g^\alpha(\underline{x}=2)$, FOR BOUNDEDLY INTELLIGENT ANTAGONISTS 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 SAMPLE $d_2 = \{(1,2;1/2),(2,2;1)\}$

α	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}
1/4	$\frac{1}{2},\frac{1}{2}$	$\frac{3}{4},\frac{1}{2}$	$\frac{3}{4},\frac{1}{2}$	$1,\frac{1}{2}$	$\frac{1}{2},\frac{3}{4}$	$\frac{3}{4},\frac{3}{4}$	$\frac{3}{4},\frac{3}{4}$	$1,\frac{3}{4}$	$\frac{1}{2},\frac{3}{4}$	$\frac{3}{4},\frac{3}{4}$	$\frac{3}{4},\frac{3}{4}$	$1,\frac{3}{4}$	$\frac{1}{2},1$	$\frac{3}{4},1$	$\frac{3}{4},1$	$1,1$
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	$\frac{1}{2},\frac{1}{2}$	$\frac{1}{2},\frac{1}{2}$	$\frac{1}{2},\frac{1}{2}$	$1,\frac{1}{2}$	$\frac{1}{2},\frac{1}{2}$	$\frac{1}{2},\frac{1}{2}$	$\frac{1}{2},\frac{1}{2}$	$1,\frac{1}{2}$	$\frac{1}{2},\frac{1}{2}$	$\frac{1}{2},\frac{1}{2}$	$\frac{1}{2},\frac{1}{2}$	$1,\frac{1}{2}$	$\frac{1}{2},1$	$\frac{1}{2},1$	$\frac{1}{2},1$	$1,1$

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 move. That analysis would parallel the approach taken in [24] used to study bandit problems, and would further increase the performance gap between the (best, worst) pair of algorithms.

VII. ROLE OF THE ANTAGONIST'S "INTELLIGENCE"

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

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

One way to approach this problem is to build the antagonist'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 moves, then there are $(|X|/\alpha|X|)$ possible subsets of joint moves of size $\alpha|X|$.[16] 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 this particular subset, $\bar{x}^*$ is estimated by selecting the best response out of the $\mathcal{S}_i$ payoff values known to the antagonist. Of course, it may be the case that there are no samples in $\mathcal{S}_i$ having the agent's move $\underline{x}$ and in that case the antagonist can only select a random response. In this case, the agent will obtain the average payoff $\sum_{\bar{x}} f(\underline{x}, \bar{x})/l(\underline{x})$. If we assume that all subsets of size $\alpha|X|$ are equally likely, then the agent's payoff function against an antagonist with bounded intelligence is given by

$$g^\alpha(\underline{x}) = \begin{pmatrix} |X| \\ \alpha|X| \end{pmatrix}^{-1} \sum_{\mathcal{S}_i \in \mathcal{S}(X, \alpha|X|)} \arg\min_{(\underline{x}, \bar{x}) \in \mathcal{S}_i} f(\underline{x}, \bar{x}).$$

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 antagonist). However, we also observe that for the same sample d_2, the average $[g(\underline{x} = 1) g(\underline{x} = 2)]$ values are $[5/8 \;\; 7/8]$ for $\alpha = 1/4, [29/48 \;\; 41/48]^\top$ for $\alpha = 2/4, [9/16 \;\; 13/16]^\top$ for $\alpha = 3/4$, and $[1/2 \;\; 3/4]^\top$ for $\alpha = 4/4$. For this sample, $d_2(a, \underline{A}_{\text{best}})$ continues to beat $(a, \underline{A}_{\text{worst}})$ by the same amount independent of α.

VIII. CONCLUSION

We have introduced a general framework for analyzing NFL issues in a variety of contexts, GO. When applied to self-play GO establishes 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 analyzes for optimization in nonself-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 $\min_{\bar{x}} f(\cdot, \bar{x})$. 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, the NFL theorems still hold.

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 multiagent systems [27].

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

$$cd_1 + e_1 c d_2 + e_2 c \bar{d}_1 \bar{d}_2 + \bar{e}_1 \bar{c} d_1 + \bar{c} d_2 + e_3 \bar{c} \bar{d}_1 \bar{d}_2$$

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 $\bar{a} = \neg a$. It is convenient to factor the Boolean expression as

$$c(d_1 + e_1 d_2 + e_2 \bar{d}_1 \bar{d}_2) + \bar{c}(\bar{e}_1 d_1 + d_2 + e_3 \bar{d}_1 \bar{d}_2).$$

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,

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

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

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	$(\lvert Y\rvert - p + 1)^{\bar{l}-1} - (\lvert Y\rvert - p)^{\bar{l}-1}$

and p with appropriate multiplicities. The multiplicities are given in Table III.

In the continuum limit this sum becomes the integral

$$\int_0^1 dj_1 \int_0^1 dj_2 \int_0^1 dk_1 P(k_1) \int_0^1 dk_2 P(k_2) \int_0^1 dn P(n) \times \int_0^1 dp P(p) \{c(d_1 + e_1 d_2 + e_2 \bar{d}_1 \bar{d}_2) + \bar{c}(\bar{e}_1 d_1 + d_2 + e_3 \bar{d}_1 \bar{d}_2)\}$$

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 nondistinct 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 nondistinct 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 Fig. 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 containing Y values at $\{1/10\} \cup d_m^{y|\underline{x}}$.[17] 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 $2/10$.

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 $\underline{x}$				*		*			*	
$P(\tilde{g}\|d_m)$	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 antagonist 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 $2.48/10$.

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 antagonist 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 $2.94/10$.

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

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

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 $\mathbb{E}(\tilde{g}|d_m) = (1/10)(8^{\underline{3}} - 4^{\underline{3}})/(3 \cdot 7^{\underline{2}}) = 52/21 \approx 2.48/10$.

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

In Fig. 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

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

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

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

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

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

Combining these results, we obtain the final result

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

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

$$\begin{aligned}\mathbb{E}(\tilde{g}|d_m) &= r\left(1 - r + \frac{1}{|Y|}\right)^b \\ &\quad + \sum_{\tilde{g}=1/|Y|}^{r-1/|Y|} \tilde{g}\left\{\left(1 - \tilde{g} + \frac{1}{|Y|}\right)^b - (1 - \tilde{g})^b\right\} \\ &= \sum_{r'=1/|Y|}^{r}\left(1 - \left(r' - \frac{1}{|Y|}\right)\right)^b\end{aligned}$$

where we have canceled 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

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

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 $(1/10)294/100 = 2.94/10$.

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

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

by the integral

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

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 $261.3/1000$.

Acknowledgment

The authors would like to thank the editor D. Fogel and the anonymous referees for many helpful comments.

References

[1] R. J. Aumann and S. Hart, *Handbook of Game Theory with Economic Applications* Amsterdam, The Netherlands, 1992.

[2] N. Barricelli, "Numerical testing of evolution theories: Part II: Preliminary tests of performance, symbiogenesis and terrestrial life," *Acta Biotheoretica*, vol. 16, no. 3,4, pp. 99–126, 1963.

[3] A. Bucci and J. B. Pollack, "Order-theoretic analysis of coevolution problems: Coevolutionary statics," in *Proc. Bird of a Feather Workshops, Int. Conf. Genetic Evol. Comput.*, A. Barry, Ed., New York, NY, 2002, pp. 229–235.

[4] K. Chellapilla and D. B. Fogel, "Evolution neural networks, games, and intelligence," *Proc. IEEE*, vol. 87, pp. 1471–1498, 1999.

[5] S. Christensen and F. Oppacher, "What can we learn from no free lunch? A first attempt to characterize the concept of a searchable function," in *Proc. Genetic Evol. Comput. Conf.*, L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H. M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, Eds., San Francisco, CA, July 7–1, 2001, pp. 1219–1226.

[6] Y. Cohen, T. L. Vincent, and J. S. Brown, "A G-function approach to fitness minima fitness maxima, evolutionarily stable strategies, and adaptive landscapes," *Evol. Ecology Res.*, vol. 1, pp. 923–942, 1999.

[7] D. W. Corne and J. D. Knowles, "No free lunch, free leftovers theorems for multiobjective optimization problems," in *Lecture Notes in Computer Science*. New York: Springer-Verlag, 2003, Proc. 2nd Int. Conf. Evol. Multi-Criterion Opt., pp. 327–341.

[8] E. D. de Jong and J. B. Pollack, "Ideal evaluation from coevolution," *Evol. Comput.*, vol. 12, no. 2, pp. 159–192, 2004.

[9] S. Droste, T. Jansen, and I. Wegener, "Optimization with randomized search heuristics—The (A)NFL theorem, realistic scenarios, and difficult functions," *Theor. Comput. Sci.*, vol. 287, no. 1, pp. 131–144, 2002.

[10] T. M. English, "Optimization is easy and learning is hard in the typical function," in *Proc. Congr. Evol. Comput.*, A. Zalzala, C. Fonseca, J. H. Kin, and A. Smith, Eds., La Jolla, CA, 2000, pp. 924–931.

[11] ——, "On the structure of sequential search: Beyond "no free lunch"," in *EvoCOP, Lecture Notes in Computer Science*, J. Gottlieb and G. R. Raidl, Eds. New York: Springer-Verlag, 2004, vol. 3004, pp. 95–103.

[12] S. L. Epstein, "Toward an ideal trainer," *Mach. Learn.*, vol. 15, pp. 251–277, 1994.

[13] S. G. Ficici and J. B. Pollack, "A game-theoretic approach to the simple coevolutionary algorithm," in *Parallel Problem Solving from Nature VI*, M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, H.-P. Schwefel, and J. J. Merelo, Eds. New York: Springer-Verlag, Sep. 2000.

[14] D. B. Fogel and A. Ghozeil, "A note on representations and variation operators," *IEEE Trans. Evol. Comput.*, vol. 1, no. 2, pp. 159–161, Jul. 1997.

[15] D. B. Fogel, T. J. Hays, S. L. Hahn, and J. Quon, "A self-learning evolutionary chess program," *Proc. IEEE*, vol. 92, pp. 1947–1954, 2004.

[16] L. J. Fogel and G. H. Burgin, "Competitive goal-seeking through evolutionary programming," Air Force Cambridge Research Labs., Cambridge, MA, Final Rep. Contract AF 19(628)-5927.

[17] A. Franz, K. H. Hoffmann, and P. Salamon, "Best possible strategy for finding ground states," *Phys. Rev. Lett.*, vol. 86, pp. 5219–5222, 2001.

[18] D. Fudenberg and J. Tirole, *Game Theory*. Cambridge, MA: MIT Press, 1991.

[19] W. D. Hillis, "Coevolving parasites improce simulated evolution as an optimization procedure," in *Artificial Life*, C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, Eds. Reading, MA: Addison-Wesley, 1992, pp. 313–322.

[20] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Cambridge, MA: MIT Press, 1975.

[21] C. Igel and M. Toussaint, "A no-free-lunch theorem for nonuniform distributions of target functions," *J. Math. Modeling, Algorithms*, vol. 3, no. 4, pp. 313–322, 2004.

[22] M. Koeppen, D. H. Wolpert, and W. G. Macready, "Remarks on a recent paper on the 'no free lunch' theorems," *IEEE Trans. Evol. Comput.*, vol. 5, no. 3, pp. 295–296, Jun. 2001.

[23] W. G. Macready and D. H. Wolpert, "What makes an optimization problem hard?," *Complexity*, vol. 5, pp. 40–46, 1996.

[24] ——, "Bandit problems and the exploration/exploitation tradeoff," *IEEE Trans. Evol. Comput.*, vol. 2, pp. 2–22, 1998.

[25] J. Maynard Smith, *Evolution and the Theory of Games*. Cambridge, U.K.: Cambridge Univ. Press, 1982.

[26] M. Oltean, "Searching for a practical evidence for the no free lunch theorems," in *BioInspired Approaches to Advanced Information Technology, LNCS 3141*, A. Ijspeert *et al.*, Eds. Lausanne, Switzerland: Springer-Verlag, Jan. 2004.

[27] S. Parsons and M. Wooldridge, "Game theory and decision theory in multi-agent systems," *Autonomous Agents Multi-Agent Sys.*, vol. 5, no. 3, pp. 243–254, Sept. 2002.

[28] N. J. Radcliffe and P. D. Surry, "Fundamental limitations on search algorithms: Evolutionary computing in perspective," *Computer Science Today: Recent Trends and Developments, LNCS 1000*, pp. 275–291, 1995.

[29] J. Reed, R. Toombs, and N. Barricelli, "Simulation of biological evolutionary and machine learning," *J. Theroet. Biol.*, vol. 17, pp. 319–342, 1967.

[30] C. Schumacher, M. D. Vose, and L. D. Whitley, "The no free lunch and problem description length," in *Proc. Genetic Evol. Comput. Conf.*, L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, Eds., 7–11, 2001, pp. 565–570.

[31] D. Whitley, "A free lunch proof for gray versus binary encodings," in *Proc. Genetic Evol. Comput. Conf.*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds., Orlando, FL, 1999, pp. 726–733.

[32] D. H. Wolpert, "The relationship between PAC and the statistical physics framework, the Bayesian framework and the VC framework," in *The Mathematics of Generalization*. Reading, MA: Addison-Wesley, 1995, pp. 117–215.

[33] ——, "The existence of a priori distinctions between learning algorithms," *Neural Comput.*, vol. 8, pp. 1391–1420, 1996.

[34] ——, "The lack of *a priori* distinctions between learning algorithms," *Neural Comput.*, vol. 8, pp. 1341–1390, 1996.

[35] ——, "Bounded rationality game theory," *Inf. Theory*, 2004, submitted for publication.

[36] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–83, Apr. 1997.

David H. Wolpert received the B.S. degree from Princeton University, Princeton, NJ, and the Ph.D. degree in physics from the University of California, Santa Barbara.

He is currently a Senior Computer Scientist at the NASA Ames Research Center, Moffett Field, CA, and a consulting Professor at Stanford University, Stanford, CA. He was formerly the head of a data mining group at IBM Almaden Research and a Postdoctoral Fellow at the Santa Fe Institute. His work primarily concerns how to design collectives of complex adaptive agents to achieve a global goal (i.e., distributed control and/or distribution optimization). Other work investigates the bounds on computation that hold in broad classes of physical universes, and the use of self-dissimilarity as a complexity measure. He also works on the extension of game theory to subrational games using techniques from statistical physics.

William G. Macready is Director of Product Development at D-Wave Systems, Vancouver, BC, Canada, a company commercializing quantum computing for optimization. At D-Wave, he is responsible for developing algorithms which exploit quantum mechanical effects in superconducting electronics. Prior to D-Wave, he was a Senior Scientist at the NASA Ames Research Center, Moffett Field, CA, where he worked on machine learning, data analysis and distributed optimization applications. His primary interests lie in applications of probabilistic inference to problems in learning, optimization, and control. He has published in a variety of fields including physics, machine learning, optimization, economics, and molecular biology.