Coevolutionary Free Lunches

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

***Abstract*— ecent work on the foundations of optimization has begun to uncover its underlying rich structure. In particular, the "No Free Lunch" (NFL) theorems [WM97] state that any two algorithms are equivalent when their performance is averaged across all possible problems. This highlights the need for exploiting problem-speci£c knowledge to achieve better than random performance. In this paper we present a general framework covering most search scenarios. In addition to the optimization scenarios addressed in the NFL results, this framework covers multi-armed bandit problems and biological-style evolution of multiple co-evolving agents. As a particular instance of the latter, it covers "self-play" problems. In these problems the agents work together to produce a champion, who then engages one or more antagonists in a subsequent multi-player game. In contrast to the traditional optimization case where the NFL results hold, we show that in self-play there are free lunches: in coevolution some algorithms have better performance than other algorithms, averaged across all possible problems. However in the typical coevolutionary scenarios encountered in biology, where there is no champion, NFL still holds.ecent work on the foundations of optimization has begun to uncover its underlying rich structure. In particular, the "No Free Lunch" (NFL) theorems [WM97] state that any two algorithms are equivalent when their performance is averaged across all possible problems. This highlights the need for exploiting problem-speci£c knowledge to achieve better than random performance. In this paper we present a general framework covering most search scenarios. In addition to the optimization scenarios addressed in the NFL results, this framework covers multi-armed bandit problems and biological-style evolution of multiple co-evolving agents. As a particular instance of the latter, it covers "self-play" problems. In these problems the agents work together to produce a champion, who then engages one or more antagonists in a subsequent multi-player game. In contrast to the traditional optimization case where the NFL results hold, we show that in self-play there are free lunches: in coevolution some algorithms have better performance than other algorithms, averaged across all possible problems. However in the typical coevolutionary scenarios encountered in biology, where there is no champion, NFL still holds.R**

I. Introduction

Optimization algorithms have proven to be valuable in almost every setting where quantitative £gures of merit are available. Recently, the mathematical foundations of optimization have begun to be uncovered [**?**], [MW96], [FHS01], [**?**], [**?**]. One particular result in this work, the "No Free Lunch" (NFL) theorems, establishes the equivalent performance of all optimization algorithms when averaged across all possible problems [**?**]. Numerous works have extended these early results, and considered their application to different types of optimization (e.g. to multi-objective optimization [**?**]). The web site www.no-free-lunch.org offers a list of recent references.

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

One important example of coevolution is "self-play", where the players cooperate to train one of them as a champion. That champion is then pitted against antagonists in a subsequent multi-player game. The goal is to train that champion to perform as well as possible in that subsequent game. (For a checkers example see [CF99].)

Coevolution can also be used for problems which on the surface appear to have no connection to a game (for an early application to sorting networks see [Hil92]). Coevolution in these cases enables escape from poor local optima in favor of better local optima. We will refer to all players other than the one of direct attention as that player's "opponents", even when (as in self-play) the players are actually cooperating.

In this paper we £rst present a mathematical framework that covers both traditional optimization and coevolutionary scenarios. It also cover other scenarios like multi-armed bandits and general systems of co-evolving agents. We then use that framework to explore the differences between traditional optimization and coevolution. We £nd dramatic differences between the two classes of problems. In particular, unlike the fundamental NFL result, there are algorithms which are superior to other algorithms for *all* problems in the self-play domain. However in the typical coevolutionary scenarios encountered in biology, where there is no champion, NFL still holds.

II. General Framework

In this section we present a general framework, and illustrate it on two examples. It is only a slight extension of the framework used in [WM97].

A. Formal framework speci£cation

Say we have two spaces, X and Z. Typically $x \in X$ will be a joint strategy followed by our player(s), and $z \in Z$ will be a probability distribution over some space of possible rewards/payoffs to the champion, or over possible £gures of merit, or some such. We also have a £tness function

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

In the most common example z is a probability distribution over rewards. Then f can be viewed as the speci£cation of an x-conditioned probability distribution of rewards.

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

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

Each $d^x(t)$ is a particular $x \in X$. Each $d^z(t)$ is a (perhaps stochastic) function of $f(d_X(t'))$. For example, say z's – values of $f(x)$ – are probability distributions over reward values. Then $d^z(t)$ could consist of the full distribution $f(d^x(t))$. Alternatively, it could consist of a moment of that distribution, or even a random sample of it. In general, we allow the function specifying $d^z(t)$ to vary with t, although that freedom will not be exploited here. As shorthand we will write $d(t)$ to mean the pair $\left(d^x(t), d^z(t)\right)$.

A search algorithm a is an initial distribution $P_1(d^x(1))$, together with a set of $m-1$ conditional distributions $P_t(d^x(t)|d_{t-1}), t = 2, \ldots m$. Such an algorithm speci£es what x to choose, based on the information uncovered so far, for any time-step t. Finally, we have a vector-valued cost function, $C(d_m, f)$ which we use to assess the performance of the algorithm. Often our goal is to £nd the a that will maximize $E(C)$.

The NFL theorems concern averages over all f of quantities involving C. For those f-averages of C to be independent of the search algorithm, it is crucial that C does not depend on f. When that independence is relaxed (as in self-play), the NFL theorems need not hold. This is how one can have free lunches. This papers explores this phenomenon.

B. Examples of the framework

a) Example 1: One example of this framework is the scenario considered in the NFL theorems. There each z is a probability distribution over a space $Y \subseteq \mathbb{R}$. For convenience we take X and Y countable. Each $d_f(t)$ is a sample of the associated function $z(t) = f(d^x(t))$. The search algorithm is constrained so that

$$P_t(d^x(t) = x|d_{t-1}) = 0\ \forall x \in d_{t-1}^x, \tag{3}$$

i.e., so that the search never revisits points already sampled.[1] Finally, $C(d, f)$ is allowed to be any scalar-valued function that depends on d exclusively.

The NFL theorems apply to any scenario meeting these speci£cations.

b) Example 2: Another example is the multi-arm bandit problem introduced for optimization by Holland [Hol75] and analyzed in [MW98]. The scenario for that problem is identical to that for the NFL results, except that there are no constraints on the search algorithm, $Y = \mathbb{R}$ and every z is a Gaussian. The fact that revisits are allowed means that NFL need not apply.

[1]This requirement is just to "normalize" algorithms. In general, an algorithm that sometimes revisits points can outperform one that never does. Our requirement simply says that we're purely focusing on how well the algorithms choose new points, not how smart they are about whether to end the search with a new point or one already visited. See [WM97].

c) Example 3: Self-play is identical to the NFL scenario except that C depends on f. This dependence is based on a function $A(d_m)$ mapping d_m to a subset of X. A selects the champion from the training period and the possible opponent responses to the champion. C uses such a function to specify the quality of the search algorithm that generated d_m as follows:

$$C(d_m, f) = \min_{x \in A(d_m)} \mathbb{E}_{f(x)}. \tag{4}$$

where $\mathbb{E}_{f(x)}$ is the expected value of the distribution of payoffs $f(x)$ obtained from x. Intuitively, this measure is the worst possible payoff to the champion.

To see in more detail how this describes self-play, assume two players, with strategy spaces X_1 and X_2, X_1 being the strategy space of our champion. Take x to be the joint strategy of our players in any particular game, i.e., $X = X_1 \times X_2$. So d_m^x speci£es the m strategies followed by our champion (as well as that of the other player) during the m training games. d_m^z is the associated set of rewards to our champion.

Let $x_1 \in X_1$ be the strategy our champion elects to follow based on that training data. That strategy can be represented as the set of all joint-strategies x whose £rst component is x_1. It is this representation that we use, by writing that set of all x's consistent with x_1 as $A(d_m)$. Say the antagonist our champion faces is able to choose the worst possible element of X_2 (as far as expected reward to our champion is concerned), given that our champion chooses strategy $A(d_m)$. If the antagonist does this the expected reward to our champion is given $C(d_m, f)$ as de£ned above. Obvious variants of this setup replace the worst-case nature of C with some alternative, have F be stochastic, etc. Whatever variant we choose, typically our goal in self-play is to choose a and/or A so as to maximize $E(C)$, the expectation being over all possible d_m. The fact that C depends on f means that NFL need not apply. The mathematical structure that replaces NFL is explored in the following sections of this paper.

d) Example 4: While the rough description of self-play looks like a special case of the more general coevolution scenario, in terms of our framework they are quite different. In the general coevolution scenario there are a total of N agents (or players, or species', or genes, etc). Their strategy spaces are written X_i, as in self-play. Now though X is extended to include the previous joint "population frequency" value, as well as the current joint strategy. Formally, we write

$$X = (X_1, u_1) \times \cdots \times (X_N, u_N), \tag{5}$$

and interpret each $u_i \in \mathbb{R}$ as agent i's previous population frequency. As explained below, the reason for this extension of X is so that a can give the sequence of joint population frequencies that accompanies the sequence of joint strategies.

In the general coevolution scenario each z is a probability distribution over the possible current population frequencies of the agents. So given our de£nition of X, we interpret f as a map taking the previous joint population frequency, together with the current joint strategy of the agents, into a probability distribution over the possible joint population frequencies of the agents.

As an example, in evolutionary game theory, the joint strategy of the agents at any given t determines how each one's population frequency changes in that time-step **WGM: what is this ref?** [**?**]. Accordingly, in the replicator dynamics of evolutionary game theory, f takes a joint strategy $x_1 \times \ldots x_N$ and the values of all agents' previous population frequencies, and based on that determines the new value of each agent's population frequency.

As before, each $d^z(t)$ contains the information coming out of $f(d^x(t))$, i.e., the set of current population frequencies. a now plays two roles. One of these is to directly incorporate those current population frequencies into the $\{u_i\}$ components of $d^x(t+1)$. The other is, as before, to determine the joint strategy $[x_1, \ldots, x_N]$ for time $t+1$. As in self-play, this strategy of each agent i is given by a (potentially stochastic and/or time-varying) function a_i. An application of a is given by the simultaneous operation of all those N distinct a_i on a common d_t, as well as the transfer of the joint population frequency from $d^z(t)$, to produce $d^x(t+1)$.

Finally, C is now a vector with N components, each component j only depending on the associated $d_t(j)$. In general in biological coevolution scenarios (e.g., evolutionary game theory), there is no notion of a champion being produced by the search and subsequently pitted against an antagonist in a "bake-off". Accordingly, there is no particular signi£cance to results for C's that depend on f.

This means that so long as we make the approximation, reasonable in real biological systems, that x's are never revisited, all of the requirements of Ex. 1 are met. This means that NFL applies. So in particular, say we restrict attention to the particular kinds of a's of evolutionary game theory. Then any two choices of a – any two sets of strategy-making rules $\{a_i\}$ – perform just as well as one another, averaged over all f's. More generally, we can consider other kinds of a as well, and the result still holds.

III. Application to Coevolution

In example 3 of section II-B we introduced a self-play model of coevolution. In the remainder of this paper we show how free lunches may arise in this setting, and quantify the a priori differences between certain coevolutionary algorithms.

In coevolution agents (or game strategies) are paired against each other in a (perhaps stochastically formed) sequence to generate a set of 2-player games. After m distinct training games between an agent and its opponents, the agent enters the competition. Performance of the agent is measured with a payoff function. The payoff function when the ith agent plays move $\underline{x}_i$ and i's opponent plays $\overline{x}_i$ is given by $f_i(\underline{x}_i, \overline{x}_i)$. If we indicate the joint move of i and its opponent as $x_i = (\underline{x}_i, \overline{x}_i)$ we can write the payoff to agent i as $f_i(x_i)$. In the following we make no assumption about the structure of moves except that they are £nite. $\underline{x}$ might represent a sequence of plays representing an entire game of checkers and $\overline{x}$ might represent a complete set of opponent responses to each play. The payoff function $f(\underline{x}, \overline{x})$ might then represent the outcome of the game as $+1$ for a win for i, 0 for a draw, and -1 for a loss. Illegal joint moves can be eliminated by appropriately limiting the space of moves and opponent responses in order to satisfy the rules of the game. In other applications, $\underline{x}$ might represent an algorithm to sort a list and $\overline{x}$ a mutable set of lists to be sorted. The payoff would then re¤ect the ability of the algorithm to sort those lists in $\overline{x}$.

We de£ne the payoff for agent i playing move $\underline{x}_i$ independent of an opponent's reply, $g(\underline{x}_i)$, as the least payoff over all possible opponent responses (a minimax criteria): $g_i(\underline{x}_i) \equiv \min_{\overline{x}_i} f_i(\underline{x}_i, \overline{x}_i)$. With this criterion, the best move an agent can make is that move which maximizes g_i 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 i's possible moves, but more generally across all joint moves of i 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 i's minimax performance g_i.

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_i for all such joint moves is better than the worst value for some other move, $\underline{x}'_i$. To do this certainly requires that m is greater than the total number of possible moves by the opponent but even for very large m unless all possible opponent responses have been explored we can not make any such guarantees.

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

In the following sections we explore this observation further.

A. De£nitions

As much as possible we follow the notation of [WM97] extending it where necessary. That paper should be consulted as motivation for the analysis framework we employ. Without loss of generality we now consider two player games, and leave the agent index i implicit. If there are $\underline{l}$ moves available to an agent, we label these by $\underline{x} \in \underline{X} \equiv [1, \cdots, \underline{l}]$. For each such move we assume the opponent may choose from one of

$\overline{l}(\underline{x})$ possible moves forming the space $\overline{X}(\underline{x})$. [2] For simplicity we will take $\overline{X}(\underline{x})$ to be independent of $\underline{x}$. Consequently, the size of the joint move space is $|X| = \sum_{\underline{x}=1}^{\underline{l}} \overline{l}(\underline{x})$. 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 won't 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.

In [WM97] a population is a sample of distinct points from the input space X and their corresponding £tness values. In this coevolutionary context the notion of a population of sampled con£gurations needs to be extended to include opponent responses. For simplicity we assume that £tness payoffs are a deterministic function of joint moves. Thus, rather than the more general output space Z, we assume payoff values lie in a £nite totally ordered space. Consequently, the £tness function is the mapping $f : X \mapsto Y$ where $X = \underline{X} \times \overline{X}$ is the space of joint moves. As in the general framework a population of size m is represented as

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

where $d_m^x(i) = \left\{ d_m^{\underline{x}}(i), d_m^{\overline{x}}(i) \right\}$ and $d_m^y(i) = f(d_m^{\underline{x}}(i), d_m^{\overline{x}}(i))$ and $i \in [1, \cdots, m]$ labels the samples taken. In the above de£nition $d_m^{\underline{x}}(i)$ is the ith move made by the agent, $d_m^{\overline{x}}(i)$ is the opponent response, and $d_m^y(i)$ is the corresponding payoff. As usual we assume that no joint con£gurations are revisited. A particular coevolutionary optimization task is speci£ed by de£ning the payoff function that is to be extremized. As discussed in [WM97] a class of problems is de£ned by specifying a probability density $P(f)$ over the space of possible payoff functions. As long as both X and Y are £nite (as they are in any computer implementation) this is straightforward.

In addition to this extended notion of a population, there is an additional consideration in the coevolutionary setting, namely the decision of what move to make in the competition based upon the results of the training population. Formally, we encapsulate the process of making this decision as $\underline{A}$.[3] $\underline{A}$ consists of a set of distributions (one for each m since we would like $\underline{A}$ to select a move regardless of the size of the training set) of the form $P(\underline{x} \in \underline{X} | d_m)$. If A deterministically returns a single move, we indicate the mapping from training population to move as $\underline{A}(d_m)$. To summarize, the de£nition of search method is extended for coevolution to include:

- A search rule a which determines the manner in which a population is expanded during training and is formally given by the set of distributions $\{P_t(d^x(t)|d_{t-1})\}_{t=1}^m$. This corresponds exactly to the de£nition of a search algorithm in [WM97] used in non-coevolutionary optimization.
- A move-choosing rule $\underline{A}$ mapping probabilistically or deterministically to the single move used in the competition. We write $\underline{A}$ explicitly as the probability density $\underline{A}(\underline{x}|d_m)$ where $\underline{x} \in \underline{X}$. For deterministic $\underline{A}$ we write the density as $\underline{A}(\underline{x}|d_m) = \delta\left(\underline{x} - \underline{A}(d_m)\right)$.

The tuple $(a, \underline{A})$ is called a search process (as opposed to a search algorithm in [WM97]).

The search process seeks a strategy that will perform well in competition. If $\underline{A}$ is deterministic the natural measure of the performance of search process $(a, \underline{A})$ obtained during training is $C = \min_{i \in [1,m]} f\left(\underline{A}(d_m), d_m^{\overline{x}}(i)\right)$. (If $\underline{A}$ is not deterministic then we use the weighted average $\sum_{\underline{x} \in \underline{X}} \min_{i \in [1,m]} f\left(\underline{x}, d_m^{\overline{x}}(i)\right) \underline{A}(\underline{x}|d_m)$.) The best $(a, \underline{A})$ for a particular f are those which maximize C.

The traditional version of NFL (for non-coevolutionary optimization) de£nes the performance differently since there is no opponent. In the simplest case the performance of a (recall that there is no choosing algorithm) might be measured as $C = \max_{i \in [1,m]} d_m^y$. One traditional NFL result states that the average performance of any pair of algorithms is identical, or formally, $\sum_f P(C|f, m, a)$ is independent of a.[4] A natural extension of this previous results considers a non-uniform average over £tness functions. In this case the quantity of interest is $\sum_f P(C|f, m, a)P(f)$ where $P(f)$ weights different £tness functions. NFL results can be proven for other non-uniform $P(f)$ [**?**].

A result akin to this one in the coevolutionary setting would state that the unform average $\sum_f P(C|f, m, a, \underline{A})$ is independent of a and $\underline{A}$. However, as we have informally seen, 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 the next section.

IV. An Intuitive Example

Before proving the existence of free lunches we give a motivating example to both illustrate the de£nitions made in the above section and to show why we might expect free lunches to exist. Consider the concrete 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. $\overline{X} = \{1, 2\}$, and there are only two possible payoff values, i.e. $Y = \{1/2, 1\}$. In this simple case there are 16 possible functions and these are listed in Table I. We can see that in this simple example the minimax criteria gives a very biased distribution over possible performance measures: 9/16 of the functions have $g = \begin{bmatrix} 1/2 & 1/2 \end{bmatrix}$, 3/16 have $g = \begin{bmatrix} 1/2 & 1 \end{bmatrix}$, 3/16 have $g = \begin{bmatrix} 1 & 1/2 \end{bmatrix}$, and 1/16 have $g = \begin{bmatrix} 1 & 1 \end{bmatrix}$ where $g = \begin{bmatrix} g(\underline{x} = 1) & g(\underline{x} = 2) \end{bmatrix}$.

If we consider a particular population, say $d_2 = \{(1, 2; 1/2), (2, 2; 1)\}$, the payoff functions that are consistent with this population are $f_9, f_{10}, f_{13}, f_{14}$ and the corresponding distribution over g functions is $1/2 \begin{bmatrix} 1/2 & 1/2 \end{bmatrix}^\top$ and $1/2 \begin{bmatrix} 1/2 & 1 \end{bmatrix}^\top$. Given the fact that any population will give a biased sample over g functions it may not surprising that there are free lunches. We might 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

[2] 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).

[3] The notation $\underline{A}$ is meant to suggest that, unlike the $A(d_m)$ function introduced earlier, $\underline{A}$ de£nes only the champions move, and not the possible responses to this move.

[4] Actually far more can be said, and the reader is encouraged to consult [WM97] for details.

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

TABLE I

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

sample of gs. In the next section we prove the existence of free lunches by constructing such a pair of algorithms.

V. PROOF OF FREE LUNCHES

In this section a proof is presented that there are free lunches for coevolutionary by constructing a pair of search processes one of which explicitly has performance equal to or better than the other for all possible payoff functions f. As in earlier NFL work we assume that both $|X|$ and $|Y|$ are £nite.[5] For convenience, and with no loss in generality, we normalize the possible Y values so that they are equal to $1/|Y|, 2/|Y|, \cdots, 1$.

The pair of processes we construct use the same search rule a (it is not important in the present context what a is) but different deterministic move choosing rules $\underline{A}$. In both cases a Bayesian estimate based on uniform $P(f)$ and the d_m at hand is made of the expected value of $g(\underline{x}) = \min_{\overline{x}} f(\underline{x},\overline{x})$ for each $\underline{x}$. Since we are striving to maximize the worst possible payoff from f, the optimal search process selects the move which maximizes this expected value while the worst process selects the move which minimizes this value. More formally, if $E(C|d_m, a, \underline{A})$ differs for the two choices of $\underline{A}$, always being higher for one of them, then $E(C|m, a, \underline{A}) = \sum_{d_m} P(d_m|a) E(C|d_m, \underline{A})$ differs for the two $\underline{A}$. In turn, $E(C|m, a, \underline{A}) = \sum_{f,C}[C \times P(C|f, m, a, \underline{A}) \times P(f)] \propto \sum_{f,C}[C \times P(C|f, m, a, \underline{A})]$ for the uniform prior $P(f)$. Since this differs for the two $\underline{A}$, so must $\sum_f P(C|f, m, a, \underline{A})$.

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 makes move $\underline{x}$ and the opponent replies. In the example of section IV 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 $P(\tilde{g}(\underline{x})|\underline{x}, d_m) = \sum_f P(\tilde{g}(\underline{x})|\underline{x}, d_m, f) P(f)$ for uniform $P(f)$. Of the entire population d_m only the subset sampled at $\underline{x}$ is relevant. We assume that there are $k(\underline{x}, d_m) \leq m$ such values.[6] 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. $r(\underline{x}, d_m) = \min_{\overline{x} \in d_m^{\underline{x}}} d_m^y(\underline{x}, \overline{x})$. Since payoff values are normalized to lie between 0 and 1, $0 < r(\underline{x}, d_m) \leq 1$. Given $k(\underline{x}, d_m)$ and $r(\underline{x}, d_m)$, $P(\tilde{g}|\underline{x}, d_m)$ is independent of $\underline{x}$ and d_m and so we indicate the desired probability as $\pi_{k,r}(\tilde{g})$.

[5]Recall that $|X| = \sum_{\underline{x}} \overline{l}(\underline{x})$.

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

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

$$\mathbb{E}(\tilde{g}(\underline{x})|\underline{x}, d_m) = \frac{1 - (1 - r(\underline{x}, d_m))^{\overline{l}(\underline{x}) - k(\underline{x}, d_m) + 1}}{\overline{l}(\underline{x}) - k(\underline{x}, d_m) + 1}. \quad (6)$$

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

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

As long as $C_m(\underline{x})$ is not constant (which will usually be the case since the r values will differ) $(a, \underline{A}_{\text{best}})$ and $(a, \underline{A}_{\text{worst}})$ will differ, and the expected performance of $\underline{A}_{\text{best}}$ will be superior. This proves that the expected performance over all payoff functions of algorithm $(a, \underline{A}_{\text{best}})$ is greater than that of algorithm $(a, \underline{A}_{\text{worst}})$.

VI. OTHER FREE LUNCHES

We have shown the existence of free lunches for coevolution by constructing a pair of algorithms with the same search rule a but different move-choosing rules $\underline{A}$ and showing different performance. Unsurprisingly, we can also construct algorithms with different expected performance but having the same move-choosing rule and different search 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 coevolutionary settings.

Take the simple case where all $\underline{l}$ agent moves offer the same possible number of opponent responses, say $\overline{l}$. Consider a search rule that explores $m = \underline{l}$ distinct joint samples. Agent moves are labelled by $\underline{x} \in \{1, \cdots \underline{l}\}$ and opponent responses are labelled by $\overline{x} \in \{1, \cdots \overline{l}\}$. For simplicity we assume that $\underline{l} = \overline{l}$.

Search rule a_1 explores the joint moves $(1,1), \cdots, (1,m)$ while search rule a_2 explores the joint moves

$(1,1),\cdots,(m,1)$, i.e. a_1 exhausts opponent responses to $\underline{x}=1$ while a_2 only samples 1 opponent response to each of it's m possible moves. For the move choosing rule we apply the Bayes optimal rule used earlier: select the move $\underline{x}$ that has the highest expected $g(\underline{x})$ value when averaged over payoff functions consistent with the observed population.

To start we determine the expected performance of an algorithm which does not have the bene£t 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. We note that for any given player 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 £rst $|Y|^{\bar{l}}$ 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 $n_{\bar{l}}(i)=\big[(i+1)/|Y|\big]^{\bar{l}}-\big[i/|Y|\big]^{\bar{l}}$. This average is 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 which samples some opponent responses has to do better than this.

Next we consider the algorithm a_1 which exhaustively explores all opponent responses to $\underline{x}=1$. There are $|Y|^{\bar{l}}$ possible populations that this algorithm might see. For each of these populations, 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 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 $\langle g\rangle$ above. Since the move choosing rule maximizes the expected value of g the expected performance of a_1 for this population is $\max(\min d_m^y,\langle g\rangle)$. Averaged over all functions (populations) 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 populations. Converting the sum over all populations into a sum over the minimum value of the population we £nd

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

If we de£ne $i_g\equiv\lceil|Y|(1-\langle g\rangle)\rceil$ 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

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

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 element of the population occurs at a different $\underline{x}$. For any given observed population the optimal move for the player is $\underline{x}^*=\arg\max d_m^{y|\underline{x}}$ (i.e. the move corresponding to the largest payoff observed in the population). With this insight, we observe that when summing over all functions, there are $|Y|^{\bar{l}-1}$ possible completions to $\max d_m^y$ 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 (populations) is

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

We proceed in the same fashion as above by de£ning[7] $i_d\equiv|Y|(1-\max d_m^y)$ (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 populations 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

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

The continuum limit in this case is found as

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

where $B(x,y)$ is the beta function de£ned 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 which does not have access to any training data.

[7]There is no need to take the ceiling since i_d is automatically an integer.

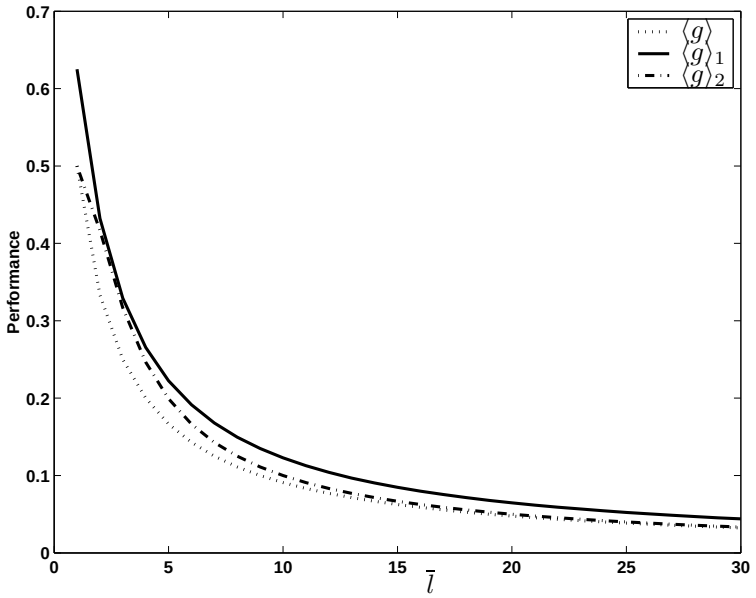

Fig. 1. Expected performance of algorithm a_1 (indicated as $\langle g \rangle_1$) which exhaustively enumerates the opponents response to a particular move, and algorithm a_2 (indicated as $\langle g \rangle_2$) which samples only 1 opponent response to each move. For comparison we also plot $\langle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

In Figure 1 we plot the expected performance of a_1 and a_2 as a function of $\overline{l}$ (recall that $m = \underline{l} = \overline{l}$). Algorithm a_1 outperforms algorithm a_2 on average for all values of $\overline{l}$.

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 $|Y|^{-\overline{l}\,\overline{l}} \sum_f \theta\big(\text{perf}_1(f) - \text{perf}_2(f)\big)$ where $\text{perf}_1(f)$ is the performance of algorithm a_1 on payoff function f, $\text{perf}_2(f)$ is the performance of algorithm a_2 on the same f, and θ is a step function de£ned 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[8]

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

Similarly, the performance of algorithm a_2 is given by

$$\text{perf}_2(f) = \min_{\overline{x}} f(\underline{x}_2^*, \overline{x}) \quad \text{where} \quad \underline{x}_2^* = \arg\max_{\underline{x}} d_m^{y|\underline{x}}.$$

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

$$\frac{j_1}{|Y|} \equiv f(1,1), \quad \frac{j_2}{|Y|} \equiv f(2,1)$$
$$\frac{k_1}{|Y|} \equiv \min_{\overline{x}}\{f(1,\overline{x}) | \overline{x} \neq 1\}, \quad \frac{k_2}{|Y|} \equiv \min_{\overline{x}}\{f(2,\overline{x}) | \overline{x} \neq 1\}$$
$$\frac{l}{|Y|} \equiv \max_{\underline{x}}\{f(\underline{x},1) | \underline{x} \neq 1,2\},$$
$$\frac{m}{|Y|} \equiv \min_{\overline{x}}\{f(\underline{x}_2^*, \overline{x}) | \underline{x}_2^* \neq 1,2, \overline{x} \neq 1\}$$

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

In the de£nition of m, $\underline{x}_2^*$ is the move chosen by a_2 if a_2 doesn't choose move $\underline{x}_2^* = 1$ or 2, the speci£c value of $\underline{x}_2^*$ is irrelevant. Given these de£nitions, the performance of the two algorithms are

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

and

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

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, l and m 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 D.

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

A. 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 suf£cient to demonstrate the availability of free lunches in coevolutionary contexts. However, we can build on these insights to construct even better (and even worse) algorithms by also determining (at least partially) the Bayes-optimal search rule, $(a, \underline{A})$, which builds out the training set, and selects the champion strategy. That analysis would parallel the approach taken in [MW98] used to study bandit problems. and would further increase the performance gap between the (best,worst) pair of algorithms.

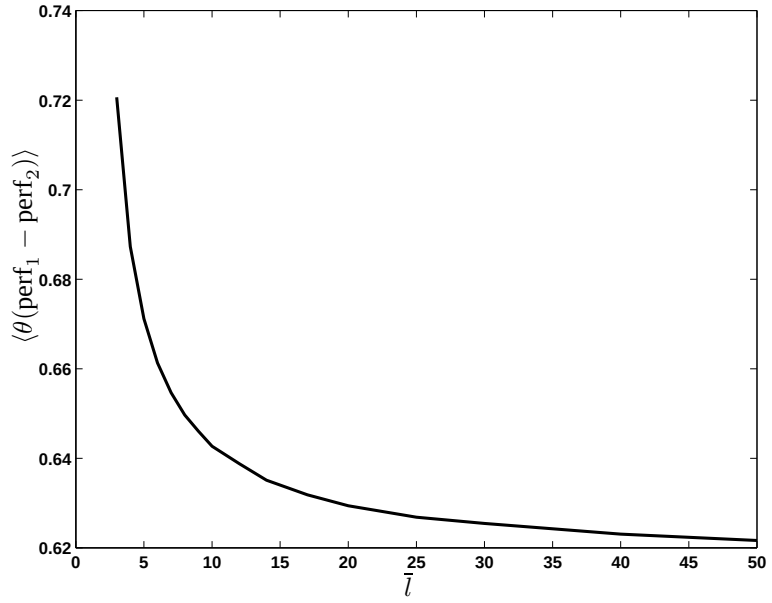

Fig. 2. The fraction of functions in the continuum limit where $\mathrm{perf}_1 \geq \mathrm{perf}_2$ The £gure was generated with 10^7 Monte Carlo samples of the integrand for each value of $\bar{l}$.

VII. The Role of Opponent "Intelligence"

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

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

One way to approach this problem is to build the opponent's bounded intelligence into the agent's payoff function g and proceed as we did in the omniscient case. If $|X|$ is the number of joint moves, then there are $\binom{|X|}{\alpha|X|}$ possible subsets of joint moves of size $\alpha|X|$.[9] 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 $\overline{x}^*$ is estimated by selecting the best response out of the $\mathcal{S}_i$ payoff values known to the opponent. 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 opponent can only select a random response. In this case the agent will on average obtain the average payoff $\sum_{\overline{x}} f(\underline{x}, \overline{x})/l(\underline{x})$. If we assume that all subsets of size $\alpha|X|$ are equally likely, then the agent's payoff function against a boundedly intelligent opponent is given by

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

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

VIII. Conclusions

We have introduced a general framework for NFL-like results in a variety of contexts. When applied to coevolution we have proven the existence of pairs of algorithms one of which is superior to another for all possible joint payoff functions f. This result stands in marked contrast to similar analyses for optimization in non-coevolutionary settings. Basically, the result arises because under a minimax criteria the sum over all payoff functions f is not equivalent to a sum over all functions $\min_{\overline{x}} f(\cdot, \overline{x})$. We have shown that for simple algorithms we can calculate expected performance over all possible payoff functions and in some cases determine the fraction of functions where one algorithm outperforms another.

As indicated by the richness of results around non-coevolutionary optimization it is clear that we have only just scratched the surface of an analysis of coevolutionary optimization. Many of the same questions posed in the non-coevolutionary can be asked in the coevolutionary setting.

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

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

TABLE II

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

Such endeavors may be particularly rewarding at this time given the current interest in the use of game theory and coevolution for multi-agent systems [**?**].

REFERENCES

[CF99] K. Chellapilla and D. B. Fogel. Evolution, neural networks, games, and intelligence. *Proc. IEEE*, 87:1471–1498, 1999.

[FHS01] A. Franz, K. H. Hoffmann, and P. Salamon. Best possible strategy for £nding ground states. *Phys. Rev. Lett.*, 86:5219 – 5222, 2001.

[Hil92] W. D. Hillis. Coevolving parasites improce simulated evolution as an optimization procedure. In C. G. Langton, C.Taylor, J. D. Farmer, and S. Rasnussen, editors, *Arti£cial Life II*, pages 313–322. Addison Wesley, 1992.

[Hol75] J. H. Holland. *Adaptation in Natural and Arti£cial Systems.* MIT Press, Cambridge, MA, 1975.

[MW96] W. G. Macready and D. H. Wolpert. What makes an optimization problem? *Complexity*, 5:40–46, 1996.

[MW98] W. G. Macready and D. H. Wolpert. Bandit problems and the exploration/exploitation tradeoff. *IEEE Trans. Evol. Comp.*, 2:2, 1998.

[WM97] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Trans. Evol. Comp.*, 1:67–83, 1997.

APPENDIX

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

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

To derive the result we generalize from a concrete example. Consider the case where $|Y| = 10$, $\overline{l}(\underline{x}) = 5$, and $k = 3$. A particular instantiation is presented in Figure 3. In this case $r = 4/10$ which is not the true minimum for responses to $\underline{x}$. The probability that r is the true minimum is simply $k/\overline{l}(\underline{x})$. 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 $\{1/10\} \cup d_m^{y|\underline{x}}$.[10] Since the total number of possibilities consistent with the data is $\binom{|Y|-k}{\overline{l}(\underline{x})-k}$ this fraction is $\binom{|Y|-k-1}{\overline{l}(\underline{x})-k-1}/\binom{|Y|-k}{\overline{l}(\underline{x})-k} = (\overline{l}(\underline{x}) - k)/(|Y| - k)$. Similarly, $P(\tilde{g} = 2/10|d_m)$ is $\binom{|Y|-k-2}{\overline{l}(\underline{x})-k-1}/\binom{|Y|-k}{\overline{l}(\underline{x})-k}$ because we know that the function can not contain a sample having £tness less than $2/10$.

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

Thus, in the general case, we have

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

where $a = |Y| - k$, $b = \overline{l}(\underline{x}) - k$, $\theta(x) = 1$ iff $x > 0$, and $\delta_{\tilde{g},r} = 1$ iff $\tilde{g} = r$. Since it is easily veri£ed 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 correctly normalized. The expected value of $\tilde{g}$ is therefore

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

Evaluating this sum we £nd

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

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

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

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

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

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

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

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

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

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

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

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

Combining these results we obtain the £nal result

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

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

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

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

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

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

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

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

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

C. Continuum Limit

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

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

by the integral

$$E(\tilde{g}|d_m) = \int_0^r dr'\ (1 - r')^b = \frac{1}{b+1}\left\{1 - (1 - r)^{b+1}\right\}. \tag{7}$$

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 is 261.65/1000 while the approximation gives 261.3/1000.

D. 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 de£ned as in Section VI.

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

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

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

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

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

j_1	1
j_2	1
k_1	$(\|Y\|-k_1+1)^{\bar{l}-1}-(\|Y\|-k_1)^{\bar{l}-1}$
k_2	$(\|Y\|-k_2+1)^{\bar{l}-1}-(\|Y\|-k_2)^{\bar{l}-1}$
l	$l^{\bar{l}-2}-(l-1)^{\bar{l}-2}$
m	$(\|Y\|-m+1)^{\bar{l}-1}-(\|Y\|-m)^{\bar{l}-1}$

TABLE III

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

In the continuum limit this sum becomes the integral

$$\int_0^1 dj_1 \int_0^1 dj_2 \int_0^1 dk_1\, P(k_1) \int_0^1 dk_2\, P(k_2) \int_0^1 dl\, P(l) \times$$
$$\int_0^1 dm\, P(m) \big\{ c(d_1 + e_1 d_2 + e_2 \overline{d}_1 \overline{d}_2) + \overline{c}(\overline{e}_1 d_1 + d_2 + e_3 \overline{d}_1 \overline{d}_2) \big\}$$

where $P(k_1) = (\bar{l}-1)(1-k_1)^{\bar{l}-2}$, $P(k_2) = (\bar{l}-1)(1-k_2)^{\bar{l}-2}$, $P(l) = (\bar{l}-2)l^{\bar{l}-3}$, $P(m) = (\bar{l}-1)(1-m)^{\bar{l}-2}$, 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 $(j_1, j_2, k_1, k_2, l, m)$ using the respective probability distributions. Samples from $P(u) = m(1-u)^{m-1}$ are obtained by sampling values v from $U(0,1)$ and transforming so that $u = 1 - v^{1/m}$; samples from $P(w) = mw^{m-1}$ are obtained via $w = v^{1/m}$.