# Black-box Search by Elimination of Fitness Functions

Gautham Anil
School of Electrical Eng. & Computer Science
University of Central Florida
Orlando, FL, USA
ganil@cs.ucf.edu

R. Paul Wiegand
Institute for Simulation & Training
University of Central Florida
Orlando, FL, USA
wiegand@ist.ucf.edu

## ABSTRACT

In black-box optimization an algorithm must solve one of many possible functions, though the precise instance is unknown. In practice, it is reasonable to assume that an algorithm designer has some basic knowledge of the problem class in order to choose appropriate methods. In traditional approaches, one focuses on how to select samples and direct search to minimize the number of function evaluations to find an optima.

As an alternative view, we consider search processes as determining which function in the problem class is the unknown target function by using samples to eliminate candidate functions from the set. We focus on the efficiency of this elimination process and construct an idealized method for optimal elimination of fitness functions. From this, we place our technique in context by relating performances of our idealized method to common search heuristics (e.g., (1+1) EA), and showing how our ideas relate to No Free Lunch theory. In our discussion, we address some of the practicalities of our method.

Though in its early stages, we believe that there is utility in search methods based on ideas from our elimination of functions method, and that our viewpoint provides promise and new insight about black-box optimization.

## Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems;
G.1.6 [**Numerical Analysis**]: Optimization

## General Terms

Theory, Algorithms, Performance

## Keywords

Black-box optimization, evolutionary algorithms, elimination of functions

## 1. INTRODUCTION

A traditional view of the optimization process of a randomized search heuristic such as evolutionary computation is one in which a sequence of samples from the search space is used to determine future samples in order to chart an efficient course toward some global optimum. Like any optimization method, it is often useful to bound the number of samples that can be expected before the optimum is found as it relates to the problem size. Indeed, there are a number of such runtime analyses for certain types of evolutionary algorithms (EAs) on particular problems (see, for example, [4, 10] and [7] for a survey).

Of course for a particular instance of an optimization problem, one typically needn't even apply an EA: if we know the precise function, we often already know the answer. E.g., If we know the domain is OneMax, as there is only one fitness function, then we know the solution is always the all-ones string and we can just return it without doing any sampling. A slightly more subtle point is that for the set of all problems, an EA is no better or worse than any other method (see NFL [11]).

Indeed, evolutionary algorithms are black-box methods: We apply them in the case where we are asked to solve one of many possible problems, though we do not know the precise instance. In fact, Droste *et al.* [3, 2] make the compelling case that our real interest is in understanding how these optimization methods perform over various *classes* of problems. The natural point here is that it is reasonable to assume that this class is not the entire space of all problems, and that an algorithm designer may have (at least) some basic knowledge of the problem class in order to chose appropriate methods.

We agree. In fact, this observation suggests an alternative view of black-box optimization methods: Rather than see these systems as using samples to find the optimum, one can turn the view around and consider search processes as using the samples to help determine which function in our problem class is the target function of the problem. This shift in perspective allows us to consider each step in the search process as a sorting activity, using samples to partition the function set into functions that can be definitively eliminated as the target problem and those that (as yet) cannot be eliminated. The entire process, then, becomes a series of decisions about how to refine our knowledge of these sets given samples from the search space, and analysis can focus on whether such algorithms make the most of the potential information available to them given knowledge of the problem class.

When one considers this view, several new questions come to light. What kind of guarantees can we make about how search methods approach this elimination activity? How do traditional methods use the information provided by their samples to implicitly eliminate functions from the function set (or do they)? We can also posit idealized methods for explicitly searching by optimal elimination of fitness functions (OEFF), and we can relate the performance of various methods to this standard on particular problem classes. Finally, we can recast existing theories in terms of OEFF.

In this paper, we introduce such a view and discuss what can be learned by adopting it. We also outline OEFF and prove that it meets the known black-box lower bound for a problem class involving a particular generalization of OneMax under idealized circumstances, then illustrate this experimentally. We then place our work in context of what is already known by first demonstrating how OEFF can be used verify the well-known NFL result for problem classes closed under permutation[8]), how the (1+1) EA and a simple variant can be viewed as performing an implicit elimination of fitness functions, and how greedy sampling affects function elimination. We conclude with a short discussion of how these ideas can be used more generally for analysis purposes, as well as suggest several ways to find practical implementations of our OEFF method.

However, as a note of caution, we do not present this technique in this form as a practical search strategy. Our goal in developing OEFF is to explore the performance bounds on black box search in the presence of perfect *a priori* information about any domain. Our analysis centers on the question: How well can (and do) algorithms leverage what knowledge is available from the problem class? We feel this model might be useful for analysis of search strategies and at best, an inspiration for designing new ones. We believe this tool provides a better understanding of the dynamics and limits of black box optimization and a clearer perspective on the process of optimization.

## 2. ELIMINATION OF FITNESS FUNCTIONS

Consider a search space $\mathcal{X}$ and a class of functions $\mathcal{F} \subset \{f : \mathcal{X} \mapsto \mathbb{R}\}$. A black-box algorithm is presented a particular target function from that class to optimize, though which function is unknown to it *a priori*. The traditional goal of such algorithms is to sample the search space and determine a search point that optimizes the unknown function. Without loss of generality, we consider maximization: given some $u \in \mathcal{F}$ find $\hat{x} = \text{argmax}_{x \in \mathcal{X}}\, u(x)$. We write a sequence of $k$ search points sampled by the algorithm as $X_k := \langle x_1, x_2, \ldots x_k \rangle$, where $X_0 := \langle\,\rangle$.

The information in such a sequence of samples can also be used to try to determine which of the functions in $\mathcal{F}$ matches the target function $u$. Information from the samples partitions the function set into those functions that remain candidate matches for the target function and those that can be definitively eliminated as the current problem instance. Indeed, it's possible to view virtually any black-box optimization algorithm as attempting to perform this kind of elimination of fitness functions (implicitly or explicitly).

With this view, we can characterize an algorithm by the process that it uses to perform this partitioning. Our framework uses the notation $R(X_k, u)$ to refer to this characterization. $R$ describes the set of functions that remain (cannot be eliminated) as candidate matches to the target functions given a sample sequence. $R(X_k, u)$ is also written as $R^{(k)}$. Different algorithms can be described by different methods of updating $R$.

### 2.1 Ideal Domain Knowledge

The major motivation that underlies our change in view is the assumption that the person designing an optimization method for the problem class (the engineer) *knows something* about that class, and that with this knowledge they have the opportunity to construct better methods.

This assumption is not unusual or unreasonable. EA practitioners construct all manners of variations of evolutionary methods in both principled and ad hoc ways in order to gain some advantage on the type of problem for which they are focused. Indeed, a well-known view of EA operators is that they define the topology connecting search points in a space, which is why some genetic operators in traditional EA methods do better on some problems and worse on others. Indeed, often practitioners perform simple parameter tuning before making a serious attempt to solve the problem. Obviously, knowing something about $\mathcal{F}$ could help offload any operator analysis — ideally, we would construct the most generally productive operators for the problem class in which we are interested.

Variations of traditional methods concentrate on the notion of determining where to best sample a search space, but one can also apply domain knowledge to help identify the target instance from the problem class. For our purposes, we consider *ideal domain knowledge* to be the complete, *a priori* knowledge of the fitness values of all search points for all functions in the function class, as well as the solutions to each of those functions. This is a fairly broad and idealistic view of the notion of "domain knowledge". Abstractly, the reader can imagine that before the function elimination process begins, the algorithm is given a vast table with as many rows as there are functions in the domain and as many columns as the size of the search space. This table already possesses pre-evaluated fitness results, from which we can easily pre-compute the correct solution for each fitness function. This concept of ideal domain knowledge allows us to study search performance under conditions of optimal information, which cannot exist without this table.

For many problem classes, the solutions to functions are implicit given the instance (e.g., Generalized OneMax class discussed later) and the solution need not be explicitly stored. Additionally, for many problem classes there are ways to implicitly assume or approximate domain knowledge. However, there are certainly problem domains for which one or both is not true, and we suffer no illusions regarding the practicality of such assumptions for real and serious implementations.

For example, consider the Travelling Salesman Problem (TSP) on $n$ vertices. Assuming $q$ possible values for each distance, there are $q^{\frac{n(n-1)}{2}}$ graphs (domain) and $(n-1)!$ cycles through the vertices (search space). Knowing the instance of the problem *does not* imply knowledge of the solution, and our conception of ideal domain knowledge insists we calculate and store, *a priori*, all the solutions to a particular graph and all the graphs for which a particular cycle is a solution. With this information, if the graph is identified, we can immediately find a solution for the same. But the question we are asking isn't whether we can legitimately have access to this information to solve TSP instances, but whether we can use this idealism as a kind of "gold standard" for comparing how real algorithms make use of domain knowledge.

## 2.2 Consistent Elimination of Functions

A variety of questions suggest themselves at this point. For example, given a particular process, can we guaranty that an algorithm's partitioning process remains *consistent*? That is, will we potentially eliminate functions at one point in the sample sequence only to re-include it as a possible candidate match later on?

*Definition 1.* A search process is *inconsistent* if $\exists l > k, \hat{f}$ such that $\hat{f} \notin R(X_k, u)$ and $\hat{f} \in R(X_l, u)$, otherwise the process is *consistent*.

We also use the phrase "remains in $R$" to mean that an algorithm never produces a partitioning later in the sampling sequence that is inconsistent with an earlier partitioning.

Addressing consistency at the simplest level is straightforward: An algorithm that simply recognizes and returns a global optima must be consistent. In fact, our view is that such an algorithm is implicitly partitioning the problem class in the sense that in the abstract we might have maintained a record of which functions in our set are still candidates and not changed the underlying algorithm at all other than to update this record each step.

Each step, our generic optimizer might partition the function set as follows. Let $\hat{v} = \max\{u\}$ and $S_s$ be a partitioning that uses a single sample to determine which functions must remain un-eliminated based on whether or not the sample is maximal.

$$\begin{aligned} & S_s(x_i, u) \\ & = \begin{cases} \{f | f \in \mathcal{F},\ f(x_i) = \hat{v}\} & \text{if } u(x_i) = \hat{v} \\ \{f | f \in \mathcal{F},\ f(x_i) \neq \hat{v}\} & \text{otherwise} \end{cases} \end{aligned}$$

We can consider $R_{\text{generic}}$ to be the (consistent) contents of our record of candidate functions after some sequence:

$$R_{\text{generic}}(X_k, u) = \bigcap_{i=1}^{k} S_s(x_i, u)$$

Assuming the algorithm cannot return a result as the optimum that is not the optimum and that it must return the optimum when found, such an algorithm must be consistent. Indeed, our notion of "consistency" is very similar to Ficici's notion of *monotonicity* of solution concepts in interactive (co-optimization) domains [5].

## 2.3 Optimal Elimination of Fitness Functions

Of course, it's clear that an algorithm that remains in $R$ is not necessarily a good optimization algorithm. Our real interest is the efficiency of a given elimination process: Have we made as much use of the the samples we have seen than we might have given ideal knowledge of the domain? Or: Might we have chosen our samples such that better partitioning would have been possible? This first question is the focus in our paper, though the second is of serious concern, as well.

From an elimination of functions point of view, the goal of any algorithm should be to eliminate as many fitness functions as possible with as few samples as is possible. Leaving aside the issue of how samples are selected, the key question is: Given $u(x)$, what is the maximum number of fitness functions that we can eliminate given a particular sequence of samples and complete knowledge of the problem class fitness space?

*Definition 2.* A function $f \in \mathcal{F}$ is *incompatible with the target function* (or just *incompatible*) given some sample search point $x_i$ if $f(x_i) \neq u(x_i)$.

A search process makes the most use of its sample sequence when it eliminates every incompatible function from $\mathcal{F}$ with every sample[1]. Note that precisely how a search process can determine that a function is incompatible depends heavily on what kind of domain knowledge exists and how it can be used.

As explained in section 2.1, we assume that all the points in $\mathcal{X}$ have been pre-evaluated at all functions in $\mathcal{F}$, and that accessing particular values is computationally much less expensive than evaluating the unknown target function. For clarity, we use the notation $V_i^{(j)}$ to mean the pre-computed value of $f_j(x_i)$, where $f_j \in \mathcal{F}$.

For practical applications this is unreasonable, but here it focuses our attention on bounding the number of opportunities an elimination algorithm has to reassess the problem class partitioning. That only occurs when a new sample is presented to the unknown target function.

*Definition 3.* A search method that explicitly or implicitly eliminates of functions from $\mathcal{F}$ is *optimal* if it's partitioning process can be described as follows:

$$\begin{aligned} R_{\text{OPT}}^{(0)} &= R_{\text{OPT}}(X_0, u) = \mathcal{F} \\ R_{\text{OPT}}^{(k)} &= R_{\text{OPT}}(X_k, u) \\ R_{\text{OPT}}^{(k+1)} &= R_{\text{OPT}}^{(k)} - \left\{ f_i | f_i \in R_{\text{OPT}}^{(k)},\ V_{k+1}^{(i)} \neq u(x_{k+1}) \right\} \end{aligned}$$

When we speak of "optimality"of elimination, we are essentially discussing the efficient use of information present in the domain itself. Informally, a method that optimally eliminates functions given some sampling strategy leverages all possible domain knowledge to make the fewest possible partitioning decisions. More specifically, let A be an optimal search process that uses the smallest possible sequence of generated samples required to report a correct solution. Further, let OPT be the elimination process just described operating with the same sequence of samples as A. Given some prefix of the sample sequence, suppose OPT determines the correct solution for some problem instances, but A's prefix is insufficient for OPT in at least one other instance. In such a case, there will be at least two functions in $R_{\text{OPT}}$ that have the same fitness values for all the samples seen so far, though they have different solutions. A will have received the same fitness value results, thus even if A reports an early potential solution from one of these two functions, A has no information to distinguish them and the final result may or may not be correct. If A could certainly have identified the correct solution, there must have been at least one difference in the fitness values seen during the search, in which case OPT will have eliminated the inconsistent function(s). In that sense, OPT cannot require more samples

[1]Stochastic functions are briefly addressed later.

than A for any instance of a problem class and is optimal in that sense.

This *optimal elimination of fitness functions* (OEFF) can be viewed in several ways, and the viewpoints themselves provide some interesting insight. For example, one useful observation is that, while the specific sample values in a given sequence certainly governs the possible efficiency of such a process, the *order* of those samples is not important. But our primary view of OEFF is as a standard for comparison with other algorithms on the basis of fitness function evaluation given ideal knowledge of the problem domain.

In essence, OEFF separates search into a generic optimal search process and the more challenging issue of sample selection. This separation allows us to analyze the effects of different sampling strategies in terms of how they make use of the domain knowledge present.

Indeed, OEFF can be used as an analytical tool to help determine general black-box complexity bounds. In the simplest case, an upper bound on a problem class for a given sampling strategy may be derived. This can permit analysis of the informative power of different sampling strategies, and provides one possible valid black-box upper bound for the problem class. We present just such an analysis in the coming section for Generalized OneMax. In addition, when a lower bound for a problem class for all sample sequences can be determined, such a bound is a valid black-box lower bound for the problem class and reflects what informational efficiency is even possible for that problem. We use this property later to demonstrate the consistency of OEFF with No Free Lunch.

Note that these bounds ignore certain real-world practicalities (e.g., memory restrictions and the challenges of accessing and searching domain knowledge). Still, as we shall see, though infeasible for realistic problems, OEFF can be implemented. We will address some of the impracticalities later.

We begin by considering an OEFF process that inspects samples $\langle x_1, x_2, \ldots \rangle$ chosen under uniform distribution from $\mathcal{X}$ with no repetition. OEFF terminates when all the fitness functions in $R_k$ share at least one solution. We assume that distinguishing between functions that have the same solution is unimportant.

## 2.4 Theoretical bound for OEFF on Generalized OneMax

Let us consider applying the OEFF on a common generalization of the OneMax:

*Definition 4.* We refer to the *generalized* OneMax problem class as the class of pseudo-Boolean problems, $f : \{0,1\}^n \mapsto \mathbb{R}$, where given some target string $\hat{x} \in \{0,1\}^n$:

$$\text{OneMax}^{(\hat{x})}(x) = n - \sum_{i=1}^{n} |x_i - \hat{x}_i|$$

There are several things to note about this problem class. First, the class is not small — there are $2^n$ possible candidate functions in $\mathcal{F}$, and each function can be uniquely identified by its solution string. Second, the running time complexity of a standard (1+1) EA on such a class is $\Theta(n \lg n)$[4]. Finally, the theoretical lower bound for any black-box algorithm is known to be $\Omega(n/\lg n)$ [1], though the authors of this proof remarked that they do not believe this to be tight.

Since we can identify a function by its solution string, we can also compute the hamming distance of each solution string to the solution string of the target function. We use this fact to construct hamming distance sets $K_d$, the partition of $\mathcal{F}$ generated by $u$ based on hamming distances:

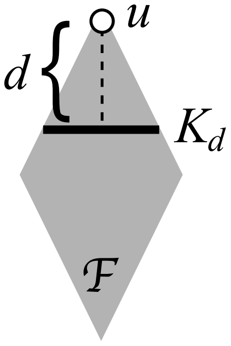

**Figure 1: We can view the set of generalized OneMax functions as a hamming space from the target function, $u$, and consider subsets of functions at a particular hamming level, $K_d$.**

$$K_d = \{f | f \in \mathcal{F},\ u(f) = d\}$$

Let's consider the question: What is the probability that a particular function will be eliminated from from hamming level, $K_d$, given some random sample $x$. First, we attempt to find the condition under which $u(x) \neq f(x)$ for $f \in K_d$ by examining the following example. Let $n = 8$, $u = 11111111$ and $f = 11110000$. Thus, $f \in K_4$. For $u(x)$ and $f(x)$ to be the same, the number of bits in $x$ that are the same as $u$ must be equal to the ones same as $f$. In the bit positions where $u$ and $f$ are identical (the first four bit positions), this is inevitable. Among the bits that are different, we need half of them to be same as $u$ and the rest $f$. E.g., if the last four bits are 0001 or 1110 etc., then $u$ and $f$ can be differentiated and $f$ will be eliminated. But if it is 0011 or 1010 etc., it is not possible to differentiate them. This example makes it clear why any $K_d$ where $d$ is odd are automatically eliminated with any sample: if $f$ has an odd number of bits equivalent in $u$, then $x$ cannot be equidistant from both $u$ and $f$.

In general, when $d$ is even, there are $\binom{d}{\frac{d}{2}}$ ways to choose half the bits where $u$ and $f$ are different. And there are $2^{n-d}$ ways to choose the bits where they are the same. Thus the probability that a sample $x$ will *not* eliminate $f \in K_d$ where $2 \leq d \leq n$ and $d$ is even ($d$ is assumed to be even from now on) is given as follows

$$P_e(d) = \frac{\binom{d}{\frac{d}{2}}.2^{n-d}}{2^n} = \frac{\binom{d}{\frac{d}{2}}}{2^d}$$

Next, we shall prove the Lemma below to simplify the composition of the fitness functions remaining after $\frac{cn}{\lg n}$ samples are made.

Lemma 1. *For a constant c, after $\frac{cn}{\lg n}$ samples the expected number of functions remaining in $K_2$ is higher than any $K_d, 2 < d \leq n$*

Proof. The expected number of samples in $K_d$ after $\frac{cn}{\ln n}$ samples is

$$\left(\frac{\binom{d}{d/2}}{2^d}\right)^{\frac{cn}{\ln n}}\binom{n}{d}$$

We have to prove, that for all $2 < d \leq n$,

$$\left(\frac{\binom{2}{2/2}}{2^2}\right)^{\frac{cn}{\ln n}}\binom{n}{2} > \left(\frac{\binom{d}{d/2}}{2^d}\right)^{\frac{cn}{\ln n}}\binom{n}{d}$$

$$\frac{1}{2}\left(\frac{1}{2}\right)^{\frac{cn}{\ln n}} > \left(\frac{\binom{d}{d/2}}{2^d}\right)^{\frac{cn}{\ln n}}\frac{(n-2)...(n-d+1)}{d!}$$

Let us consider the base of the first term on the R.H.S. Applying Sterling's approximation for factorials, we get

$$\begin{aligned}\frac{\binom{d}{d/2}}{2^d} &= \frac{d!}{\frac{d}{2}!\frac{d}{2}!2^d}\\ &\approx \frac{e^{d\ln d - d + \frac{\ln d}{2} + \frac{\ln 2\pi}{2}}}{e^{d\ln\frac{d}{2} - d + \ln\frac{d}{2} + \ln 2\pi + d\ln 2}}\\ &= e^{\frac{\ln d - 2\ln\frac{d}{2} - \ln 2\pi}{2}}\\ &= e^{\frac{\ln\frac{4d}{d^2 2\pi}}{2}}\\ &= \sqrt{\frac{2}{\pi d}}\end{aligned}$$

Using this result, we need to prove that

$$\frac{1}{2}\left(\sqrt{\frac{\frac{1}{4}}{\frac{2}{\pi d}}}\right)^{\frac{cn}{\ln n}} = \left(\frac{\pi d}{8}\right)^{\frac{cn}{2\ln n}} > \frac{(n-2)\dots(n-d+1)}{d!}$$

As R.H.S is smaller than $\left(\frac{n}{d}\right)^d$, ignoring constant term,

$$\left(\frac{\pi d}{8}\right)^{\frac{cn}{2\ln n}} > \left(\frac{n}{d}\right)^d$$

For $d = 4$, we find

$$\frac{\pi d}{8} > 1.5 > 1$$

Hence, for large enough $n$ and an adequate $c$, the L.H.S will become greater than the R.H.S. This holds true when $d$ is thought of as an integer for any $d$ where $2 < d \leq n$ as the L.H.S is exponential while the R.H.S is polynomial.

When $d$ is thought of as a function of $n$, the R.H.S grows slower than $2^n$ for any value of $d$ where $2 < d \leq n$. Consider the function $\left(\frac{n}{q}\right)^{\frac{1}{2\ln n}}$. For $q = 1$, it is a constant value $\sqrt{e} > 1$. For $q > 1$, the function approaches $\sqrt{e}$ asymptotically. Here, $q > \frac{8}{\pi}$. Picking an arbitrary value $1 < b < \sqrt{e}$, we must prove for sufficiently large $n$,

$$\left(\left(\frac{n}{q}\right)^{\frac{1}{2\ln n}}\right)^{cn} > b^{cn} > 2^n$$

This is true for $c > \frac{\ln 2}{\ln b}$. Thus the inequlity holds for any $d$ where $2 < d \leq n$. This proves the lemma. □

THEOREM 1. *For the domain of generalized* ONEMAX *of size n, the expected number of samples of the target function u required by OEFF to eliminate all incompatible functions is* $O(n/\lg n)$.

PROOF. We will determine that asymptotically almost surely there are no functions left in $K_2$ after $\frac{cn}{\lg n}$ samples. First, note that the probability of a random sample eliminating a function in $K_2$ is $\frac{\binom{2}{1}}{2^2} = \frac{1}{2}$, and the probability that it will not be eliminated is also $\frac{1}{2}$. The probability that it will be eliminated after $\frac{cn}{\lg n}$ samples is $1 - \left(\frac{1}{2}\right)^{\frac{cn}{\lg n}}$.

There are $\binom{n}{2}$ functions in $K_2$ before elimination. The probability that all these functions will be eliminated after $\frac{cn}{\lg n}$ is as follows:

$$\left(1 - \frac{1}{2^{\frac{cn}{\lg n}}}\right)^{\frac{n(n-1)}{2}}$$

From Lemma 1, we know that no $K_d$ is less likely to be empty than $K_2$, and we examine the pessimistic process where all levels are as difficult as $K_2$. Thus the probability of there being no functions other than the solution is at least:

$$P_E = \left(\left(1 - \frac{1}{2^{\frac{cn}{\lg n}}}\right)^{\frac{n(n-1)}{2}}\right)^{\frac{n}{2}} = \left(1 - \frac{1}{2^{\frac{cn}{\lg n}}}\right)^{\frac{n^2(n-1)}{4}}$$

Given that $n^4 < 2^{\frac{cn}{\lg n}}$ and $n^3 > \frac{n^2(n-1)}{4}$ for sufficiently large $n$, we know:

$$\begin{aligned}P_E &> \left(1 - \frac{1}{n^4}\right)^{n^3}\\ &= \left(\left(1 - \frac{1}{n^4}\right)^{n^4}\right)^{\frac{1}{n}}\\ &= \left(\left(1 - \frac{1}{n^4}\right)^{n^4-1}\left(1 - \frac{1}{n^4}\right)\right)^{\frac{1}{n}}\\ &\geq \left(\frac{1}{e}\left(1 - \frac{1}{n^4}\right)\right)^{\frac{1}{n}}\end{aligned}$$

Since this asymptotically approaches 1, so does $P_E$. □

## 2.5 Example elimination process

We examine a simple optimal elimination process using random sampling.

Consider the fitness function set $\mathcal{F} = \{f_0, ..., f_9\}$ over a domain $\mathcal{X} = \{y_0, ..., y_4\}$. Let the fitness value table be as given in Table 1. Given $u$ as the unknown target function, we consider the following example.

Suppose the first random sample is $y_3$ and $u(y_3) = 2$. From the table, we see that only $f_4$ and $f_8$ satisfy this criterion, so $\{f_0, f_1, f_2, f_3, f_5, f_6, f_7, f_9\}$ are eliminated (8 functions) and $\{f_4, f_8\}$ remain (2 functions). Suppose the next sample is $y_4$ and $u(y_4) = 3$, which does not allow us to eliminate an additional function. Our third sample is $y_1$, $u(y_1) = 3$. As only $f_8$ gives $y_0$ a fitness value of 8, we conclude that $u$ is $f_8$ whose solution we know to be $y_0$.

Note how the solution is not something we sampled during the search process. We will revisit this example later to examine effects of ordering of samples.

## 2.6 Applying OEFF to Generalized OneMax

To illustrate OEFF in action, as well as to get a better understanding of the actual number of samples required, we implemented OEFF and applied it to $\text{OneMax}^{(\hat{x})}$ over different problem sizes. For simplicity and clear comparison with the theoretical results, we maintain our idealism in the OEFF implementation: there is an indicator array for the complete function set to represent $R$. Additionally, the algorithm is permitted to consider the pre-evaluated values for each sample without cost. We count only evaluations of the target function. The implementation of the simplified algorithm is shown in Algorithm 1. As stated, we discuss more practical ideas for implementation later.

**Algorithm 1** OEFF(v,n) on $\text{OneMax}^{(\hat{x})}$

```
$N \leftarrow 2^n$
$c \leftarrow 0$
boolean $R[N] \leftarrow$ [true, ..., true]
RSize $\leftarrow N$
target $\leftarrow 0$
while not found do
    $x \leftarrow$ choose uniformly from $\{0,1\}^n$
    $y \leftarrow \text{OneMax}^{(\hat{x})}$ ;  $c \leftarrow c+1$
    RSize $\leftarrow 0$
    for $j = 0$ to $N-1$ do
        $R[j] = R[j] \wedge (V_x^{(j)} = y)$
        if R[j] then
            RSize $\leftarrow$ RSize +1
            target $\leftarrow$ j
        end if
    end for
end while
Return $c$,  target
```

Our experiment consisted of 20 different problem sizes ($n \in [1, 21]$). For each size, we ran the program for 1000 times. In each run, a random target function was selected from $\mathcal{F}$ by simply drawing $\hat{x}$ from $\{0,1\}^n$ uniformly at random. Thus instances from problem classes with small $n$ were over-sampled, and the sampling became increasingly sparse with the size of the problem. Each run we counted the number of times the unknown target function was evaluated before it is uniquely identified from $\mathcal{F}$, and this count was averaged across the 1000 trials for that problem size. The results are graphed in Figure 2.

In addition to the empirical values, we fitted the curve $g(n) = n/\lg n + 2\lg n - 2.25$ to the sample data. The empirical values suggest a close adherence to the theoretical bound of $O(n/\lg n)$, though a direct comparison with the bound established in [4] is not entirely fair given our optimistic assumptions.

Notice, there are other ways we might have used domain knowledge to help us partition the function set. For example, we might have used a much smaller, dispersed set of exemplar functions and used the transitive nature of the hamming distance measure to eliminate whole equivalence classes with a sample without the need to evaluate every function.

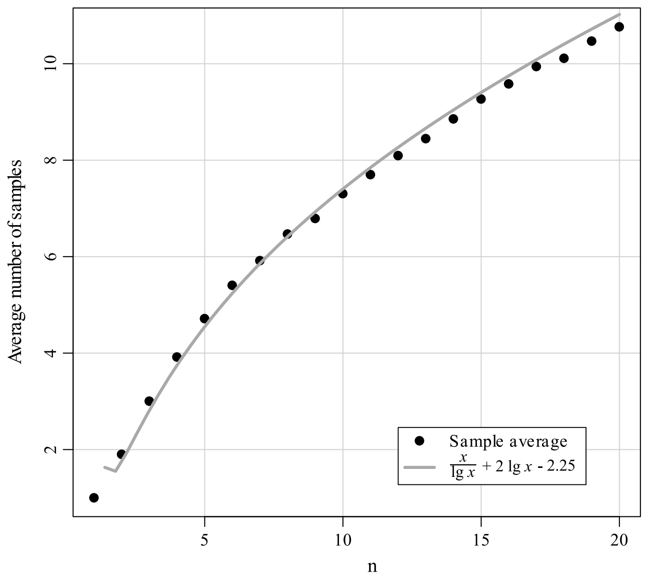

**Figure 2: The points represent mean number of evaluations before the target function is identified for problem classes of varying sizes. A fitted curve is shown for comparison with the theoretical model.**

## 3. OEFF AND EXISTING ANALYSIS

While the elimination of fitness functions view is different from traditional approaches, it is nevertheless complementary. Here we show how our knowledge of existing analysis of black-box optimization can be see through the lens of fitness function elimination.

### 3.1 OEFF and No Free Lunch

To show that OEFF is consistent with NFL, we discuss the condition in which its performance is identical to random search. Let us apply OEFF on a $\mathcal{F}$ closed under permutation. Let $V_{\mathcal{F}}$ be the list of unordered fitness values (with repetition) that are present in any fitness function in $\mathcal{F}$. $u$ is an unknown fitness function from $\mathcal{F}$. Let $\hat{v}$ be the largest value in $V_{\mathcal{F}}$, which is also the value of the global optima for *all* functions in $\mathcal{F}$. For convenience, here we also write our $X_k$ tuple as an unordered set, $X_k := \{x_1, \ldots x_k\}$.

We will consider the case where the samples taken so far do not have the fitness value $\hat{v}$. Let $\hat{v}$ be repeated $c$ times in $V_{\mathcal{F}}$. That leaves those fitness functions containing $c$ number of $\hat{v}$'s mapped to $\mathcal{X} - \{x_1, \ldots, x_k\}$ search points. As $\mathcal{F}$ is closed under permutation, these maps are also closed under permutation. Thus $R^{(k)}_{\text{OPT}}$ consists of fitness functions that can map any search point in $\mathcal{X} - \{x_1, \ldots, x_k\}$ to $\hat{v}$. Moreover, due to being closed under permutation, the number of fitness functions that map any one of those search points to $\hat{v}$ in the same as any other one.

$$\forall (x_a, x_b) \in (X - \{x_1, \ldots, x_k\})$$
$$|\{f | f \in \mathcal{F},\ f(x_a) = \hat{v}\}| = |\{f | f \in \mathcal{F},\ f(x_b) = \hat{v}\}|$$

This means that irrespective of the contents $\langle x_1, \ldots, x_k \rangle$, all search points in $\mathcal{X} - \{x_1, \ldots, x_k\}$ are equally likely to be solutions to $u$. The points have to be sampled until there are only $c$ points left. Therefore, finding their fitness is unnecessary. As this analysis is for the optimal algorithm, it is clear that it is impossible for any algorithm to solve the problem with less than $|\mathcal{X}|-c$ samples within this case. It is also clear that no algorithm including random search needs to sample more than $|\mathcal{X}| - c$ search points. Thus the performance in this case is identical to random search.

An algorithm will finish sooner if it samples the solution point (i.e., one with fitness $\hat{v}$); however, as mentioned before, for all $k$, all unsampled phenoypes are equally likely to be the solution. Inductively, we see that all permutations of search points are equally likely to result in sampling the solution. This again results in performance identical to random search.

This is, of course, the very point of the proof offered in [8] for No Free Lunch. Additionally, it's clear that our notion of function class is no different than that discussed in [6], and that their result that the number of $\mathcal{F}$ that are closed under permutation is negligible in the space of all classes is the same here, as well.

### 3.2 Function elimination in (1+1) EA on Generalized OneMax

To support our earlier argument that evolutionary algorithms tend to perform approximate implicit elimination of fitness functions, we consider (1+1) EA in the context of fitness function elimination.

The (1+1) EA (with uniform mutation operator), due to its random nature, does not exclude fitness functions or corresponding solutions strictly. However, based on the sample search point in the current population, there exists an implicit probability distribution that reflects the algorithm's estimate of the solution. Every change in the population, which occurs only after sampling a higher fitness value, results in a different and possibly better estimate of the solution.

Despite the stochasticity, in the case of a (1+1) EA working on a generalized OneMax problem, it is possible to identify a set of solutions/fitness functions $R_{(1+1)\text{ EA}}$ such that the solution to $u$ lies within this set. Again, we consider $\hat{v}$ the fitness value of the solution.

In the case of the (1+1) EA, implicit partitioning comes from two sources. First, as we've already discussed, any algorithm that is capable of recognizing and returning the solution implicitly eliminates the functions that do not correspond with the target's notion of what is or is not a solution:

$$
\begin{aligned}
&S_s(x_i, u) \\
&\quad = \begin{cases} \{f | f \in \mathcal{F},\ f(x_i) = \hat{v}\} & \text{if } u(x_i) = \hat{v} \\ \{f | f \in \mathcal{F},\ f(x_i) \neq \hat{v}\} & \text{otherwise} \end{cases}
\end{aligned}
$$

However, the (1+1) EA *also compares* values, and the correspondence between the comparison result of the target function and other functions in $\mathcal{F}$ can also be seen as a kind of implicit partitioning:

$$
\begin{aligned}
&S_c(x_i, x_j, c) \\
&\quad = \begin{cases} \{f | f \in \mathcal{F},\ f(x_i) < f(x_j)\} & \text{if } u(x_i) < u(x_j) \\ \mathcal{F} & \text{otherwise} \end{cases}
\end{aligned}
$$

With these two pieces in place, we write the elimination function for the (1+1) EA as follows.

$$
\begin{aligned}
&R_{(1+1)\text{ EA}}\ (X_k, u) \\
&\quad = \left( \bigcap_{i=1}^{k-1} S_c\left(x_i, x_{i+1}, u\right)\right) \bigcap \left( \bigcap_{i=1}^{k} S_s(x_i, u) \right)
\end{aligned}
$$

$S_s$ consists of the set of fitness functions that have $x_i$ as a solution if it recieves fitness value of a solution. It consists of everything else otherwise. $S_c$ consists of all the fitness functions that satisfy the increase in fitness value (1+1) EA dictates. Intersected with the working set at every step, $S_s$ eliminates function at a slow rate (only those function with $x_i$ as the solution) as long as $x_i$ is not a solution. But if it is, then $S_s$ eliminates enough functions to allow termination of the search. $S_c$ on the other hand eliminates functions much quicker as many more functions are likely to violate the inequality condition as a result of any sample.

From this several things are stand out. First, the elimination process here is clearly and unsurprisingly consistent. The EA will remain in $R_{(1+1)\text{ EA}}$ because the evolutionary algorithm never chooses a sample with a lower fitness value. The next population (if it changes) *will* have a higher fitness value, so only those fitness functions that do not satisfy that criterion do not survive. Second the main source of implicit partitioning here is the comparison from selection which is based on domain knowledge. While this is obvious, it underscores one of our themes: Domain knowledge *is* being applied here. Specifically, the algorithm designer has assumed that the best way to sort through the possibilities in $\mathcal{F}$ is by examining the fitness relationship between the samples one has seen. This domain knowledge provides more partitioning opportunities in the OneMax$^{(\hat{x})}$ class than our "generic" process described earlier in the paper, for example — though it's obviously not making full use of what is known by the structure of this space.

In spite of this observation, it's clear our description is overly pessimistic for a number of reasons. The EA *doesn't* sample randomly, but *directs* its sampling via the interaction between selection and mutation. To bound the number of target function evaluations of the (1+1) EA for a given problem class using our framework would require some extension to deal with the stochastic effects of the mutation operator. This is unnecessary here, because the (1+1) EA converges on a solution with significantly less number of samples than this elimination equation along with a random sampling scheme suggests. This is the result of the (1+1) EA operator's ability to increase the likelyhood of getting a sample with a higher fitness value compared to the random sampling scheme.

## 4. DIRECTING SAMPLES IN EFF

Until this point, we've considered processes in which the samples are drawn uniformly at random, independent of the search process itself. As just stated, most search heuristics use the search process to guide how samples are taken in the future. Here we take a brief look at what OEFF tells us about two such mechanisms.

### 4.1 Exact function elimination in BiRLS

One observation about generalized OneMax for application by a traditional (1+1) EA is that a simple line search should do much better. That is, if we known every bit position is independent, we needn't waste mutation effort on positions we've already fixed. Indeed, there is a quite obvious and natural way to incorporate some domain knowledge about this problem class into an EA by modifying the mutation operator as follows.

Instead of flipping uniformly, our operator will flip bits one at a time. In addition, it will not flip the same bit again. It is clear that with this operator, (1+1) EA achieves $O(n)$ for OneMax$^{(\hat{x})}$. Without loss of generality, we initialize our starting individual at $\{0\}^n$. We call this algorithm *Bit-Independent Random Local Search* (BiRLS).

We will use $b_i \in \mathcal{X} = \{0,1\}^n$ to be a binary string with all bits set to 0 except the bit in position $i$, which is set to 1. Given $\hat{v} = \max\{f\}$ for some $f \in \mathcal{F}$, we are interested in when $(\hat{v}$ AND $b_i) \neq 0^n$. For expediency and clarity, we use the short-hand notation $[f \otimes b_i]$ for this. Essentially, it is an expression that evaluates to a logical *true* if the $i$th bit of the solution of $f$ is a 1.

To model our specialized mutation, we define mask sequence as follows $M_k := \langle b_1, b_2, \ldots b_k \rangle$, though clearly the order does not matter. We define $X_k$ in the following way. The initial sample is initialized to the all zero string, $x_1 := 0^n$. Afterward, $x_i := x_{i-1}$ XOR $b_{i-1}$.

We define two separator functions:

$$\begin{aligned} S_1(b_j, x_i, x_j, u) &= \{f | f \in \mathcal{F},\ [f \otimes b_j] \wedge (u(x_i) > u(x_j))\} \\ S_2(b_j, x_i, x_j, u) &= \{f | f \in \mathcal{F},\ \neg[f \otimes b_j] \wedge (u(x_i) < u(x_j))\} \end{aligned}$$

These two functions describe the following partitioning. Given two samples, include all the functions from $\mathcal{F}$ where their solution contains the $i$th bit if that target function evaluates $u(b_i) > u(b_j)$ in $S_1$, and include all functions from $\mathcal{F}$ where their solution does not contain $i$th bit if that target function evaluates $u(b_i) < u(b_j)$ in $S_2$. That is, if flipping the bit improved fitness, then keep all functions that have it, if it did not improve it then keep all functions that did not.

With this, BiRLS can be written as an elimination function as follows.

$$\begin{aligned} & R_{\text{BiRLS}}(X_k, u) \\ &= R^{(k)}_{\text{BiRLS}} \\ &= R^{(k-1)}_{\text{BiRLS}} \cap S_s(x_k, u) \cap \\ &\quad (S_1(b_{k-1}, x_k, x_{k-1}, u) \cup S_2(b_{k-1}, x_k, x_{k-1}, u)) \end{aligned}$$

Here the elimination method is elicited taking into consideration directed sampling. So, in contrast to OEFF, the process depends extensively on the composition of the solutions. OEFF, due to its ability to look up known fitness values, can rely on fitness values alone. Neither the (1+1) EA nor our BiRLS have that luxury, and instead must rely on the composition to perform elimination.

If OEFF were presented with the sample sequence generated by BiRLS, it could not do better on generalized OneMax than $O(n)$ because opportunities to exclude functions whose solutions have bits not yet tested will not have presented themselves. So in this sense, given the sample sequence, BiRLS is optimal in its implicit function elimination process.

The reason for the difference between the linear bound and the one proved in Theorem 1, then, is only the way in which the samples are chosen. As it turns out (and perhaps counterintuitively) a bit-by-bit comparison in this case is wasteful since a great deal more than one hamming level can be removed at one time, and scanning each position in the string in some sense minimizes what can be done by any algorithm on this problem class. More generally, there's *still more* information here that was not exploited — namely the transitive relationship of the hamming distance measure that exists in *both* the search space *and* the function set space.

|  | $y_0$ | $y_1$ | $y_2$ | $y_3$ | $y_4$ |
|---|---|---|---|---|---|
| $f_0$ | 1 | 1 | 1 | 1 | 1 |
| $f_1$ | 2 | 1 | 2 | 1 | 2 |
| $f_2$ | 3 | 1 | 3 | 1 | 4 |
| $f_3$ | 1 | 2 | 1 | 1 | 1 |
| $f_4$ | 2 | 2 | 1 | 2 | 3 |
| $f_5$ | 3 | 3 | 1 | 1 | 1 |
| $f_6$ | 1 | 2 | 1 | 3 | 1 |
| $f_7$ | 2 | 3 | 1 | 1 | 2 |
| $f_8$ | 4 | 3 | 2 | 2 | 3 |
| $f_9$ | 4 | 3 | 3 | 3 | 4 |

**Table 1: A fitness function set of size 10 over a search space with five points. The information theoretically optimal choice of sample is not optimal.**

## 4.2 Greedy sample selection is not Optimal

In the above example, we noted that the method of selecting sample points can impact the efficiency that can be achieved in terms of function elimination. Though it is not our main focus, we would like to extend this discussion somewhat to help connect with a more traditional black-box view, where directing sample selection is a chief concern. Here we consider applying a simple greedy heuristic to select the sample point in our elimination of functions point of view.

Our heuristic tries to maximize the expected information gain on choosing an unsampled member of $\mathcal{X}$. We measure this using an estimate of the informational entropy [9] of the partition introduced by the sampling. We revise our earlier notation $V_{\mathcal{F}}$ to mean the set of all distinct fitness values returned by functions in $\mathcal{F}$. Let $H(x)$ be the entropy of the partition on sampling $x \in \mathcal{X}$.

$$H(x_{k+1}) = \sum_{v \in V_{\mathcal{F}}} s \ln s, \quad \text{where } \frac{|\{f | f \in R^{(k)}_{OPT} \wedge f(x) = v\}|}{|R^{(k)}_{OPT}|}$$

Maximizing entropy, we choose $x_{k+1}$ as follows.

$$x_{k+1} = \operatorname*{argmax}_{x \in \mathcal{X}} H(x)$$

As it turns out, maximizing the information gain of the next sample, *even with* complete knowledge of the $\mathcal{F} \times \mathcal{X}$ value space, does not guarantee that the number of expected samples of the unknown target function is minimal. We demonstrate this by counter example.

Let us revisit the example in Section 2.5 by apply this heuristic to Table 1. To recap, we use $y_i \in \mathcal{X}$ to refer to the specific points in the search space, and we use $x_i \in \mathcal{X}$ to refer to the points in our sample sequence (the subscripts of $y$ index the search space, the subscripts of $x$ index the sequence).

The sizes of various resulting sets generated by sampling $\mathcal{X}$ and their entropies are:

| | |
|---|---|
| Partition sizes on $y_0 : [3, 3, 2, 2]$ | $H(y_0) = 1.366$ |
| Partition sizes on $y_1 : [3, 3, 4]$ | $H(y_1) = 1.089$ |
| Partition sizes on $y_2 : [6, 2, 2]$ | $H(y_2) = 0.950$ |
| Partition sizes on $y_3 : [6, 2, 2]$ | $H(y_3) = 0.950$ |
| Partition sizes on $y_4 : [4, 2, 2, 2]$ | $H(y_4) = 1.332$ |

These equations clearly recommend $y_0$ as the optimal choice as the first sample. We need at least another sample in any case before we can terminate. Now, let $f(x_1) = f(y_0) = 1$. $R^{(1)}_{\text{OPT}} = \{f_0, f_3, f_6\}$. Now, we see that no choice of a second sample will partition $R^{(1)}_{\text{OPT}}$ into sets of size 1. Hence, a third sample will be necessary.

Instead let $x_1$ be $y_1$. The expected information gain is significantly less than the previous case. If we get $f(x_1) = v$, then by choosing $x_2 = y_{1+v}$ (the example problem class is designed such that this method of selecting the next sample point is correct), we can always reduce the size of $R^{(2)}$ to 1. Thus no more than 2 samples will be necessary. This is clearly less than the number of samples required if the heuristic is followed. Hence, the information gain is an insufficient method to guarantee the best possible sample sequence for OEFF, though it may still be useful in practice.

## 5. DISCUSSION

This paper has focused on a novel algorithmic framework for performing black-box optimization, and in so doing has taken several views. Here we offer two discussions on the two most important: 1. Using OEFF as a analytical tool for studying the efficiency of existing methods, and 2. Using OEFF (or ideas from OEFF) directly for actual optimization purposes. We offer some discussion points on both of these views below.

### 5.1 OEFF as an analytical tool

Primarily, we feel OEFF is best seen as a tool for studying problems rather than solving them. Being optimal, it can give insight into the performance bounds of classes of problems. The main idea is to establish a yardstick to see how efficiently optimization methods use the domain knowledge available (in principle, at least).

For instance, it is clear that with sufficient domain knowledge OEFF can achieve the theoretical bound on generalized OneMax. While it may be unfair to claim $O(n/\log n)$ in the general black-box scenario, it can at least be confirmed that when one has perfect domain knowledge of the problem class, the bound is tight.

In addition to analysis, OEFF may provide insight into how similar performances can be achieved in practice. The impracticality of OEFF can also be seen as an advantage since we can use it to underscore the limitations of real systems. For example, we believe OEFF can help explain why performance differences between OEFF and various practical algorithms are inevitable (i.e., the algorithms make strictly less use of existing domain knowledge).

The technique of OEFF is not specifically designed for any particular problem. However, it is able to achieve optimal performance for any $\mathcal{F}$. The key to this is the fact that though OEFF is generic, it uses very specific and complete information about $\mathcal{F}$. This information is provided in the table of fitness values that is efficiently accessible to OEFF. Practical search algorithms, lacking this information, must exploit fitness values, as well as the relationships between composition of search points to devise a heuristic to solve $u$.

The OEFF achieves optimal performance in $\text{OneMax}^{(\hat{x})}$ by sampling the unknown fitness function at random points. Without attempting broad generalizations, we feel this suggests that the sampling method may not have as much influence on the search performance as more traditional black-box optimization views imply. That is not to say that performance improvements cannot be achieved with a different sample sequence. However design engineers will clearly need to look farther than a naïve greedy heuristic for choosing samples.

From the way OEFF works, it is clear that this technique depends on variation and reliability of the fitness values. For example, if all fitness functions in $\mathcal{F}$ produced unique fitness values for a single known phenotype, and if they were exactly repeatable, then any search involves exactly one evaluation.

Notably, this technique does not depend on the connectivity of the search space effected through a combination of the genotype, operators and any genotype-phenotype map. The fitness landscape as induced by such topologies is neither relevant nor required. It relies only on the table of fitness values. This simplifies analysis by focusing our attention on the problem class rather than the details of a particular algorithm. An interesting corollary to this is that OEFF is not affected by local optima because it climbs no hills.

As for disadvantages, the important one is that fitness values of functions in $\mathcal{F}$ have to be available, making it impractical (as is) for many realistic problems.

But even this disadvantage provides useful information. Consider the general case of any $\mathcal{F}$ and OEFF, as well as our idealized domain knowledge about $\mathcal{F}$ in the pre-evaluated table. If we remove even one value from this table, say $(x_i, f_j)$, then OEFF only loses partitioning opportunities in the case where $x_i$ is drawn as a sample. Since OEFF makes maximal use of available information, it stands to reason that for any technique to be optimal in search performance in general, it has to use all the pertaining information, implicitly or explicitly. This informal argument is like a dual for the NFL. In other words, absence of any information results in no performance increase and maximum performance increase is only possible with perfect information. Again, there is no requirement that the information must be made available in the same form i.e., as a table.

Furthermore, we remind the reader that our view is predicated on the notion that, if we correctly identify the target function, we know the optimum. While this is trivial for generalized OneMax, it may be much more difficult for other problems. Nevertheless, note that for OEFF, finding the solution to the fitness functions given complete fitness values is trivial.

An important requirement for the OEFF described above is that the fitness functions in $\mathcal{F}$ cannot be stochastic to any degree. Stochastic fitness functions lead to non repeatable fitness values and consequently incorrect elimination. One solution is to make the elimination as conservative as necessary. This can be achieved by using two different yet complimentary measures. The first is to avoid eliminating fitness functions that have fitness values within an acceptable range of the fitness value given by $u$. Another is to require more than one difference in the fitness values before they are eliminated. Both measures reflect a conservative approach to elimination. The disadvantage is that this would result in significant slowdown in the search process. This is expected as the OEFF depends on the reliability of the fitness functions for performing optimal elimination.

Further restrictions are placed on $\mathcal{F}$ that it must be finite. The reason this is not an issue in population based techniques is because an approximate model of $\mathcal{F}$ lies in the imagination of the algorithm designer. This way, reasoning about the structure and shape of the landscape effected by an operator on the functions in $\mathcal{F}$ can be done even if $\mathcal{F}$ is infinite. This is not currently possible for OEFF.

## 5.2 Using EFF for practical problems

While not the primary goal, we shall discuss ways in which we might be able to adapt EFF for practical use. There are two approaches to do so. The first is a generic one that involves approximating the functioning of OEFF at the cost of optimality. The other is the possiblity of a problem specific implementation of OEFF that might retain optimality.

An important hurdle in using OEFF directly is the fact that computing the fitness function table is practically impossible. A simple approximation is to use a subset of $\mathcal{F}$ as $R^{(0)}$. We can follow this up with conservative elimination. However, this step is likely costs us accuracy of solution and even optimality of performance. The accuracy of solution increases with size of $R^{(0)}$. Moreover, we might consider dynamically changing the contents of $R^{(0)}$, retroactively applying elimination given existing samples; however, the worst case problem classes will result eventually in considering all of $\mathcal{F}$. Thus there exists a tradeoff between memory required, CPU required, and accuracy of solution.

Nevertheless, this might still be a practical option. The effect of this approximation is similar to that of evolutionary operators that search through large spaces by sampling it somewhat evenly. The solution will not be exact, but it would be available in a practical amount of time. Still, such approximation methods may complicate our assumption about inferring the solution from the target function.

The best problems for application of this approximation are those with a substantial performance difference between an unknown fitness function and a known fitness function. To explain how this might occur, consider the case of generalized OneMax. Both the known and the unknown fitness functions use the same mechanism for generating the fitness value viz. hamming distance to an internal bit string. The algorithm is not aware of $u$'s string but $u$ is. Hence, both $u$ and $f \in \mathcal{F}$ take the same time to compute fitness values.

But the requirement for exactly similar fitness does not imply identical mechanisms for generating that fitness. Consider an unknown $u$ that does not have an internal bitstring. Instead, using some very time-consuming computations, it generates fitness values identical to generalized OneMax. This mechanism, not being hamming distance, is different from the method used to generate fitness values for the known set. In this case, for solving $u$, which involves identifying its implicit internal bitstring, it might be favorable to use EFF and minimize the number of samples of $u$ necessary to do so. Here, the cost (one-time if using a table) of evaluating many known fitness functions may be more than compensated by avoiding a few evaluations of the unknown fitness function. To generalize, what we need is a set of fitness functions that can generate the fitness values similar to those of the unknown, but much faster. The concept not very dissimilar to that of a model and can be viewed as a procedural codification of an engineer's domain knowledge.

As mentioned earlier in the section, it is possible to store the contents of the fitness table implicitly. Implicit storage could allow the resulting program to eliminate potentially similar to OEFF but under real world conditions. Designing this algorithm, involves coming up with a way to implicitly maintain the set of possible fitness functions and effect an elimination process similar to OEFF without access to the fitness value tables. The mechanism used by the modified (1+1) EA to get $O(n)$ is demonstrative of how this can be achieved.

## 6. FUTURE WORK

Generalized OneMax may be considered a uninteresting benchmark by many. This is partly due to the fact that the behaviour of various evolutionary algorithms on it has been analyzed rigorously in the past. We chose it as our testing problem because it helps provide context with such previous work. It is also simple and easy to understand, and there is a known theoretical lower bound for its black-box complexity.

Next, we plan to study the behavior of OEFF on a wider variety of problems with characteristics such as dependencies between bits in the solution, deceptive fitness functions, etc. We can also study the relation between solution accuracy, solving speed and degree of fitness value stochasticity in stochastic fitness functions. More importantly, we are interested in exploring ways to approximate the OEFF for more practical implementations of this method.

Our chief interest, though, is in exploring how to efficiently leverage knowledge about a domain space in order to construct custom elimination methods equivalent to OEFF without using an explicit table of values. In particular, using known structural properties of a problem class may permit an EFF to implicitly eliminate entire subsets of functions, without directly evaluating all of them.

## 7. REFERENCES

[1] S. Droste, T. Jansen, and I. Tinnefeld, K. Wegener. A new framework for the valuation of algorithms for black-box optimization. In *Foundations of Genetic Algorithms VII*, pages 253–270. Morgan Kaufmann, 2003.

[2] S. Droste, T. Jansen, and I. Wegener. Perhaps not a free lunch, but at least a free appetizer. In *Proceedings for the 1999 Genetic and Evolutionary Computation Conference*, pages 833–839, 1999.

[3] Stefan Droste, Stefan Droste, Thomas Jansen, Thomas Jansen, Ingo Wegener, and Ingo Wegener. Upper and lower bounds for randomized search heuristics in black-box optimization. *Electronic Colloquium on Computational Complexity (ECCC*, 48:2003, 2003.

[4] Stefan Droste, Thomas Jansen, and Ingo Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276:51–81, 2002.

[5] Sevan G. Ficici. *Solution Concepts in Coevolutionary Algorithms*. PhD thesis, Brandeis University, Boston, MA, 2004.

[6] C. Igel and M. Toussaint. On classes of functions for which no free lunch results hold. *Information Processing Letters*, 86(6):317–321, 2003.

[7] P.S. Oliveto, Jun He, and X. Yao. Time complexity of evolutionary algorithms for combinatorial optimization: A decand of results. *International Journal of Automation and Computing*, 04(3), 2007.

[8] C. Schuhmacher, M.D. Vose, and L.D. Whitley. The no free lunch and description length. In *Proceedings for the 2001 Genetic and Evolutionary Computation Conference*, 2001.

[9] C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948.

[10] Ingo Wegener. Theoretical aspects of evolutionary algorithms. In *28th International Colloquium of Automata, Languages and Programming (ICALP 2001)*, pages 64–78. Springer, 2001.

[11] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 46:35–57, 1997.