Domain Specific Analysis and Modeling of Optimal Elimination of Fitness Functions with Optimal Sampling

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

This paper extends previous work that presented an algorithm called Optimal Elimination of Fitness Functions (OEFF). OEFF is by itself conditionally optimal over all problem classes, albeit impractical. Here, we complement this algorithm with an optimal sample selection strategy that removes the condition. Consequently, the performance of this combined algorithm over a domain is the black-box complexity of that domain, providing a new technique for deriving black-box complexity.

Additionally, we suggest techniques to perform runtime analysis of our extended OEFF algorithm. We discuss how those techniques can be used to build an algorithm that is targeted, practical, yet equivalent to OEFF with optimal sampling over the target domain. This is demonstrated on the Generalized Leading Ones problem domain, where we derive black-box complexity and develop an optimal algorithm.

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, leading ones, elimination of functions

1. INTRODUCTION

Randomized search heuristics such as evolutionary algorithms (EAs) are often applied quite generally to a wide variety of problem classes. It's tempting to think of such methods as relatively problem independent, but of course this is not really so. For example, EAs that use binary representations typically employ operators such as bit-flip mutation that introduce very specific domain assumptions, and they obviously perform better on some problems than on others. Given a specific problem instance or well-defined class of problems, runtime analysis [7] can be used to obtain performance bounds in terms of measures such as the expected number of evaluations before the optimum is first sampled or the probability of success within a given number of evaluations. Still, it's often useful to understand what kind of performance is *possible* on a given problem class in order to have a context for such analysis.

Indeed, randomized search heuristics such as EAs can be seen as a subset of (unrestricted) *black-box optimization* methods — those in which a design engineer is permitted to know the problem class *a priori* but not the instance, and in which optimization consists of sampling the unknown function repeatedly until a solution (optimum) can be identified. In some cases, it is possible to compute the black-box complexity [4] of some class of problems in terms of the number of function evaluations of the target function that are sufficient and necessary for a black-box algorithm to unambiguously identify an optimum. While the No Free Lunch theorem [10] dims naïve hopes of obtaining reasonable performance bounds when an algorithm contains or is given no information about the problem class, in general *most* problem classes have some kind of problem structure that can be exploited [9].

Because most existing algorithms implicitly embed their domain knowledge within the process of the algorithm itself, it's not always clear exactly how or what is being exploited (or isn't). With this in mind, [1] constructs a method that explicitly separates the general process of search dynamic, sample selection, and domain knowledge. They describe a general search method, *Optimal Elimination of Fitness Functions* (OEFF), that is optimal over any problem class given some sample sequence and *complete domain knowledge*. Optimality in their case is expressed in terms of the required number of evaluations of the unknown target function. The method can be used to gain insight about black-box search or to construct black-box complexity bounds for certain problem classes.

There are a number of unexplored questions with their approach. For example, [1] considers *random* sequences of samples, whereas clearly some sample sequences are better than others for certain problems. Additionally, the algorithm described is an analytical foil and not at all realistic, requiring exponential space and time to manage "domain

GECCO'11, July 12–16, 2011, Dublin, Ireland.

knowledge". Recognized but not implemented in that work is the observation that when there are regularities to exploit in the problem class, the domain knowledge required by OEFF for that class may be managed more efficiently with reduced space and time requirements. Indeed, for some problem classes there exist simple, practical random search heuristics that behave *equivalently* with OEFF given an optimal sample sequence. Showing that such an algorithm exists is tantamount to demonstrating the unrestricted black-box complexity for some problem class. When it is possible to elicit such an algorithm, we have reason to argue that that algorithm is an optimal black-box optimizer.

This paper addresses these issues in two key ways. First, we extend the analytical framework suggested by OEFF by describing a general and straightforward method for selecting samples optimally. Second, we demonstrate for a particular, highly structured problem class (a generalization of LEADINGONES) that there exists a simple, realistic algorithmic process that behaves equivalently to OEFF using optimal sample selection. Our bounds are consistent with known black-box bounds for the class we consider [4]. The result in this case demonstrates the power and utility of an analytical approach that conceptualizes the general search dynamic, sample selection, and domain knowledge separately. Not only can one implement an optimal algorithm for this class, but when approached in this way it becomes clear that this optimal GENERALIZEDLEADINGONES search method embeds *all* relevant domain knowledge into the search process itself.

In the next section, we explain the background for this work in more detail, including some notational preliminaries, a discussion of a variety of approaches to investigating the black-box complexity of problem classes, OEFF, and our optimal sample selection for OEFF. In Section 3, we discuss how OEFF with optimal sample selection can be modeled in order to show equivalences. The penultimate section provides our analysis of GENERALIZEDLEADINGONES, in which an optimal algorithm for this class is presented. Finally, we will discuss our conclusions and future work.

2. BACKGROUND

2.1 Notational Preliminaries

This work considers optimization of functions that map from some discrete *sample space* S into real values. When maximizing, the goal is then to find some $\hat{s}$ such that $\hat{s} = \text{argmax}_{s \in S} f(s)$, where $f : S \mapsto \mathbb{R}$, and likewise for minimization we look for $\hat{s} = \text{argmin}_{s \in S} f(s)$. We refer to the situation in which we are asked to find the optimum of a specific, unknown function as a *problem instance* and denote the unknown function as u. We use the term *problem class* to describe a known, discrete set of functions $F := \{f_1, f_2, \ldots, f_{|F|}\}$, where $|F|$ describes the cardinality of the function set.

We consider the *complete domain knowledge* of some class to be the mapping $\langle F, S\rangle : F \times S \mapsto \mathbb{R}$ where $\forall f \in F, \forall s \in S, \langle F, S\rangle(f, s) = f(s)$. This complete domain knowledge is essentially a vast table with as many rows as there are functions in the function set, as many columns as there are samples in the sample space, and in which each cell represents the objective value for a given function applied to a given sample.

We use the notation $\langle F_i, S_j\rangle$ where $S_j \subseteq S$ and $F_i \subseteq F$ to indicate a stage in the middle of the search process where we may have gained more information about the unknown function being optimized than we had in the beginning of the process. Sometimes shorthands such as F_i may be used for $\langle F_i, S_j\rangle$ for simplicity where appropriate.

Although most of the discussion in this paper assumes only that the search space S is finite, our example focuses on a problem class in which the search domain is the set of binary strings of length n. In such cases when s is a sequence of bits, we use the notation $s[i \ldots j]$ to indicate a subsequence of s starting from $i > 0$ up to and including j. Further, in the problem class of interest to this paper, all functions have a unique solution, and $f[i \ldots j]$ indicates a subsequence of the solution of f. Used alone, $[\ldots]$ indicates an ordered sequence whose contents can be accessed by index just like s.

Lastly, S_u indicates the subset of unsampled search points, and the u in the subscript should not be confused with the target function, $u : S \mapsto \mathbb{R}$.

2.2 Black-Box Complexity

In *black-box optimization*, randomized search heuristics must optimize some unknown target function from a known class of functions. The algorithm may *sample* the unknown target function by selecting elements from the sample space according to some distribution, compute the objective value of this sample on the unknown target function $u(s)$, update the probability distribution of the solution over the sample space, and repeat the process. Such a framework encompasses a wide variety of algorithms, including EAs. One may be interested in a number of performance factors in black-box optimization, but in this paper we focus on the expected number of times the unknown target function must be sampled before the global optimum is first discovered (the expected *first-hitting time*).

While ignorance of which instance we are solving is certainly a hindrance, clearly knowledge of the class of problems can be quite informative. For example, if we know that our problem class is the set of all linear functions of binary strings, then we know that probing each bit one at a time is sufficient to solve any problem. In other words, we can conclude that no algorithm that *knows* it is solving some instance from the class of linear pseudo-Boolean functions should require more than $n + 1$ function evaluations to find the optimum, where n is the length of the string. From this observation, it's natural to consider the complexity of problem classes themselves. In particular, we consider the *black-box complexity* of a problem class to be the expected first-hitting time of the *worst case* of the *best* algorithm for that problem class [3].

This black-box scenario is an *unrestricted* case in the sense that there are no constraints on *how* samples are selected or how search is directed by those samples. The class of algorithms that meet this constraint is quite broad, and it's reasonable to observe that most randomized search heuristics used in practice fall into a much narrower spectrum. Moreover, existing black-box complexity bounds for certain classes of problems suggest more optimism than performance bounds for specific algorithms on the same class allow. For example, the expected first-hitting time for the $(1 + 1)$ EA on linear functions with non-zero weights is $\Theta(n \log n)$ [3], whereas (as just observed) the black-box complexity for that class has an upper bound of $n + 1$. Another example is that of a generalized version of LEADINGONES, where the

function rewards strings for the length of prefix positions that precisely match the prefix of an unknown target string. The black-box complexity for this class is bounded above by $n/2 + o(n)$ and below by $n/2 - o(n)$ [3], while most EAs that use only bit-flip mutation will require at least $\Omega(n \log n)$ evaluations, and the $(1+1)$ EA has an expected first-hitting time of $\Theta(n^2)$ [3].

Because of this, it is sometimes useful to consider a narrower black-box scenario. For example, [6] observe that many EAs employ only some form of mutation, and thus a broad class of algorithms may be established in which we permit only those that employ operators that produce new sample points in ways that are symmetric with respect to the position of a gene. They call this scenario *unbiased* black-box complexity and produce results for some problem classes on an even narrower subset of such algorithms in which an algorithm's search operator can make use of information from at most one other previously sampled point to choose the next sample point (*unary* operators). This idea is generalized to unbiased black-box complexity given algorithms that employ k-ary operators by [2], and from these notions one can clearly obtain bounds on problem class complexity that compare more meaningfully to results for many EAs. For example, [6] shows us that the unary unbiased black-box complexity of generalized LeadingOnes is $\Theta(n^2)$ and [2] shows us how this biased problem complexity drops to $O(n \log n)$ when algorithms are permitted to use k-ary operators for $k > 1$.

Still, our interest is in examining what is optimistically *possible* (and not possible) given free access to the information in the problem class itself. Since restrictions on how samples can be selected by black-box search heuristics translate directly into restrictions on what kind of information the algorithms can make use of from the domain, unbiased black-box optimization is not appropriate for our study. It's worth noting that a variety of known EAs do not meet these bias restrictions, such as decompositional approaches like cooperative coevolutionary algorithms [8] or approaches that use explicitly asymmetric mutation operators [5]. Moreover, the optimal algorithm described at the end of this paper for solving the GeneralizedLeadingOnes problem is a simple randomized search heuristic in the spirit of most EAs and is *not* an unbiased black-box algorithm as defined by [6].

2.3 Optimal Elimination of Fitness Functions

The construction of most randomized optimization methods involve identifying and using domain knowledge to embed strategies and heuristics that leverage that knowledge. Given this, it is intuitively natural to form the perspective that the search algorithm itself must sacrifice performance on a wider problem class in order to increase performance on some target problem classes. But another view admits the possibility that search dynamics can be governed by a quite general algorithm, and domain knowledge and sample selection treated explicitly rather than as integrally a part of the search algorithm itself. Indeed, [1] introduce the idea of such a general search algorithm that is *optimal* over all problem classes: Optimal Elimination of Fitness Functions(Oeff). They specifically describe a simplified version of Oeff for problem classes that contain only functions with unique optima.

Oeff on the other hand stipulates that the *complete* domain knowledge about the target domain be made available in a specific form before execution. It then achieves optimality over the target domain by using this data *fully* during execution to identify the solution of the unknown fitness function.

Viewed this way, Oeff is no different from a very specialized algorithm as both concepts have similar access to domain knowledge. The difference lies in the fact that Oeff processes this data during runtime while the specialized algorithm already contains processed domain knowledge in algorithmic form.

The form in which Oeff accepts this domain knowledge is similar to a table of fitness values. This table has as many rows as there are fitness functions in the target problem class and as many columns as there are sample points. The fitness value that is assigned by each fitness function to each sample point is present in the corresponding cell. All the fitness values in these cells form the *complete* domain knowledge of the problem class. Despite a total lack of insightful heuristics for searching through this domain, the domain knowledge is complete because all those heuristics can be mined from this domain knowledge.

Oeff is given as Algorithm 1.

Algorithm 1 Oeff(F,S,u)

```
k ← |F|
boolean[|S|] E ← [false, ..., false]
while k > 1 do {Eliminate until only one left}
  x ← NextSample()
  u_temp ← u(x)
  for i = 1 to |F| do
    if E[i] = false then
      f ← F[i]
      if f(x) ≠ u_temp then
        E[i] ← true
        k ← k − 1
      end if
    end if
  end for
end while
for i = 1 to |F| do {Identify survivor}
  if E[i] = false then
    f ← F[i]
  end if
end for
{Find its solution using the domain knowledge}
u_max ← 0
for j = 1 to |S| do
  f_temp ← f(S[j])
  if f_temp > u_max then
    u_max ← f_temp
    u_soln ← S[j]
  end if
end for
return u_soln
```

The first loop in the algorithm is the main step, where Oeff eliminates fitness functions from the problem class. The program starts with no fitness function eliminated. First, Oeff samples the unknown fitness function based on a sample selection logic. It then compares that fitness value with the fitness value assigned by every un-eliminated fitness function in the problem class to that sample. Every fitness func-

tion that assigns a different value is eliminated. A new sample is made. This process repeats until only one fitness function remains un-eliminated as it necessarily must. In cases of problem classes where a solution may be shared by more than one fitness function, to achieve optimality, we must stop and return the solution when all un-eliminated fitness functions have the same solution. This case is not covered in Algorithm 1 for simplicity as we are concerned with a domain where all fitness functions have unique solutions. Once there is only one fitness function that has not been eliminated, this must be the unknown fitness function. We then proceed to identify the solution of this un-eliminated fitness function using the table. This solution is returned as the solution to the unknown fitness function.

Algorithm 1 requires only simple changes to make it optimal even for the cases where solutions are not unique among the objective functions. The only change required is that at the end of the elimination stage, the solutions of un-elimiated algorithms must be checked to see if all of them share a solution. If they do, present that solution immediately instead waiting until the function can be uniquely identified. But we will ignore this variation as the problem class we are interested in has unique solution.

It is clear that Algorithm 1 is exponential in both space and time. Nevertheless, we believe it is impossible to write an algorithm that takes a time that is polynomial on the number of dimensions of the search space to process each sample while being optimal over all problem classes. This is due to the runtime requirement to process the domain knowledge for each sample whose size is exponential on the dimensions of the search space.

A notable missing piece of the above algorithm is the logic of the procedure NextSample. In general, Oeff can use any sequence of samples. The analysis presented in [1] assumes Oeff uses a random sample sequence, and all comparisons between algorithms are made in terms of the *given* sequence.

2.4 Optimal Sampling for Oeff

In this section, we introduce a method of selecting samples for Oeff in a way that would minimize the expected number of future samples.An obvious method is straightforward brute force. We can conceptualize this method by approaching Oeff as a game with samples and the unknown fitness function becoming branching points on the game tree. Then, from the position on the game tree we are in, one chooses the branch that would lead to the lowest expected distance to the leaf nodes. The tree is shown in Figure 1 and the formal description follows.

Let $T_{\langle F_i,S_j\rangle}$ be a game tree induced by some sets $F_i \subseteq F$ and $S_j \subseteq S$. This tree consists of two kinds of vertices called *f-set* vertices labeled $\langle F_i, S_j\rangle$ and *s-set* vertices labeled $\langle F_i, S_j, s_k\rangle$ where $s_k \in S_j$.

The root of $T_{\langle F_i,S_j\rangle}$ is the f-set vertex labeled $\langle F_i, S_j\rangle$. Its immediate children are s-set vertices $\forall s_k \in S_j, \langle F_i, S_j, s_k\rangle$. The children of s-set vertex are f-set vertices labeled $\forall u \in F_i, \langle \text{OeffStep}(F_i, s_k, u), S_j - \{s_k\}\rangle$. The leaf nodes are the f-sets where the first member of the label is a singleton set.

Note that the condition about the singleton set applies only because the problem class has unique solutions for its members. Otherwise, if we are using the variation of Oeff that checks for common solutions, sets where all its members have common solutions would also be valid leaf nodes.

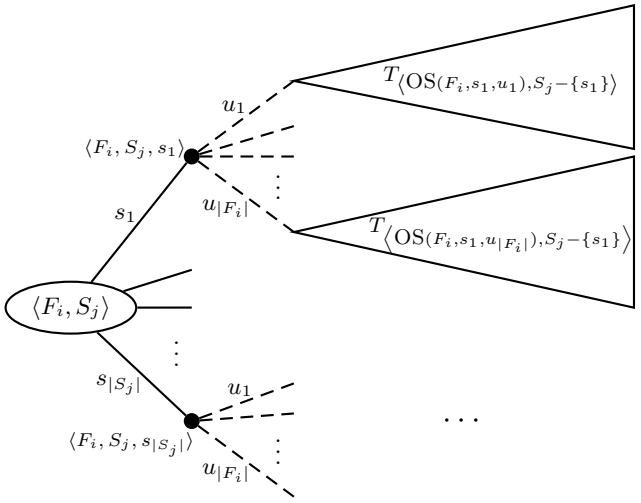

Figure 1: Illustration of game tree conceptualization of Oeff and sample selection. Here OS refers to the function $\text{OeffStep}(F, s, u) := \{f | u(s) = f(s), f \in F\}$.

However, this difference will not negatively affect the utility of the analytical techniques that are presented.

The function OeffStep, given a set F of fitness functions, a sample point s and the unknown fitness function u, returns $\{f | f \in F, f(s) = u(s)\}$.

In this game tree, there exists an edge from an f-set for every possible sample Oeff can make leading to an s-set. Similarly, from an s-set $\langle F_i, S_j, s_k\rangle$, there exists an edge for every fitness function in F_i, leading back to an f-set. Figure 1 illustrates a portion of this tree.

The optimal sample selection algorithm is given as Algorithm 2. This algorithm is given the current set of un-eliminated fitness functions F and the set of currently unsampled points S_u, and it returns a sample point that will minimize the expected number of future samples.

Algorithm 2 NextOptimalSample(F,S_u)

```
if |F| = 1 then
  return [0, false]
end if
l_min = |S_u|
for all s in S_u do
  l ← 0
  for all u in F do
    F_temp ← OeffStep(F, s, u)
    l = l + (NextOptimalSample(F_temp, S_u − {s})[0] + 1) / |F|
  end for
  if l < l_min then
    l_min ← l
    s_min ← s
  end if
end for
return [l_min, s_min]
```

This algorithm explicitly searches $T_{\langle F,S_u\rangle}$ looking for the best sample. It is recursive, calculating average tree depth at the second-level s-sets and then the root f-set after calling

itself recursively to calculate depth at the third level f-sets. The tree depth at a leaf node is 0. The tree depth at an s-set vertex is one more than the average of the child f-set vertices. The tree depth at a non-leaf f-set vertex is the minimum of the tree-depths of its child s-set vertices, and so on. This definition of tree depth aids us as it calculates the minimum expected number of samples OEFF will need before it terminates.

This sample selection algorithm is optimal in the sense that it calculates the tree depths exactly and suggests the sample at the second level with the lowest tree depth. However, it is obvious that this technique is exponential in both space and time. Nevertheless, substantially more efficient optimal sampling algorithms are unlikely since OEFF must process the exponentially large domain knowledge for every sample.

As noted before, we are interested in more than just presenting the sample selection algorithm. We are interested in how these two algorithms together can be used to create optimal algorithms for certain problem classes.

3. MODELING OEFF AND OPTIMAL SAMPLING

Next, we examine how we can use OEFF and optimal sampling to create algorithms for specific problem classes. The idea behind OEFF and optimal sampling is that if we know the domain we are working on and that we are interested in optimality over just that domain, then we can make use of this information at design time rather than runtime. We can study the behavior of the pair of algorithms on this specific table and possibly write a new algorithm with improved time and/or space complexity that mimics the behavior of OEFF and optimal sampling on that table.

The problem we will focus on is the previously presented Generalized Leading Ones. But before we start, let us examine a couple of techniques that will help us simplify OEFF and optimal sampling for a specific domain.

3.1 State Reduction

The first simplification is **State Reduction**. Note how in Figure 1, the number of edges leading to child nodes from an s-set vertex are numbered up to $|F|$. However, different edges may lead to the same f-set that are duplicated in the tree. This is because given a particular sample, different unknown fitness functions can produce the same un-eliminated set. In fact,

OBSERVATION 1. *If for $u \in F$, $F_u = \text{OEFFSTEP}(F, s, u)$, then $\forall v \in F_u, F_u = \text{OEFFSTEP}(F, s, v)$.*

Thus, for many function classes, once we have access to the table, there is opportunity for eliminating a large number of potential nodes from the game tree. It is easy to see that the reduction in states is increased when more fitness functions share fitness values. For example, in the domain of Generalized Leading Ones (see Section 4), half of all fitness values are 0, half of all the rest are 1 and so on.

Also, there is another important property that helps with state reduction. Let S_u be the set of unsampled points.

THEOREM 1. *The tree depth from $\langle F_i, S_u\rangle$ is independent of S_u.*

PROOF. Let us assume the contrary. Let $S_u, S'_u \subseteq S$ such that the tree depth of $\langle F_i, S_u\rangle$ and $\langle F_i, S'_u\rangle$ are different. This means there exists a $u \in F_i$ such that the path taken by OEFF from $\langle F_i, S_u\rangle$ to $\langle\{u\}, S_1\rangle$ is w.l.o.g shorter than the one from $\langle F_i, S'_u\rangle$ to $\langle\{u\}, S_2\rangle$ for some $S_1, S_2 \subset S_u$. Let X be the set of samples from the shorter path. For the other path, OEFF would not have chosen a longer path if X could have eliminated $F_i - \{u\}$. We must conclude that the ability of X to eliminate $F_i - \{u\}$ depends on more than u, F_i and X itself. This is not true as $f \in F - \{u\}$ will be eliminated if $\exists s \in X$ s.t. $f(s) \neq u(s)$. Thus OEFF always takes the same number of samples from $\langle F_i, S_u\rangle$ to $\langle\{u\}, S_3\rangle$ for any $S_3 \subset S_u \subseteq S$. □

Sample selection searches through the game tree described before and thus is concerned with unsampled points. But OEFF only uses tree depth during execution. Thus while modeling OEFF, one is free to ignore previous samples and thus reduce the number of states.

3.2 Symmetry

The other technique we use for modeling OEFF and sample selection, particularly the latter, is the issue of **symmetry of samples**.

As seen before, in the general case, the sample selection algorithm needs to have access to the tree depth from each sample to determine the best sample. However, in some cases, it does not matter what the actual depth of the subtree is because the depth of all possible subtrees is the same. Thus the choice of sample is immaterial as one sample is not better than any other. This eliminates the need to calculate the tree depth to achieve optimality. This can hold even if all samples do not have the same tree depth as we just need to identify a set of samples with the same lowest tree depth. Then the optimal sample selection strategy would be to randomly select a sample from this set.

This is formalized below.

Definition 1. Two sets of fitness functions F_1 and F_2 over a sample space S are said to be *permutable* with each other if a one-to-one mapping ψ from $S_1 \subseteq S$ to $S_2 \subseteq S$ exist such that,

$$\forall f \in F_1, \exists g \in F_2 \text{ s.t. } \forall s \in S_1, f(s) = g(\psi(s))$$

We will denote this as $F_1 \longleftrightarrow_{S_1,S_2} F_2$. We can also say f permutes to g to express the one-to-one mapping between F_1 and F_2.

LEMMA 1. *If $F_1 \longleftrightarrow_{S_1,S_2} F_2$, and if S_1 and S_2 consists of all samples that may be selected for sampling for F_1 and F_2 respectively, then F_1 and F_2 has the same tree depth.*

PROOF. We will prove the two trees isomorphic by induction. Consider $T_{\langle F_1,S_1\rangle}$ and $T_{\langle F_2,S_2\rangle}$. We know that for the two root f-set vertices $\langle F_1, S_1\rangle$ and $\langle F_2, S_2\rangle$, $\forall s \in S_1, \psi(s) \in S_2$ and vice-versa and $F_1 \longleftrightarrow_{S_1,S_2} F_2$.

Let $\langle F_i, S_i\rangle$ be a vertex in $T_{\langle F_1,S_1\rangle}$ and $\langle F'_i, S'_j\rangle$ be a vertex in $T_{\langle F_2,S_2\rangle}$ such that $F_i \longleftrightarrow_{S_i,S'_i} F'_i$. Consider the child f-set vertices $\langle\text{OEFFSTEP}(F_i, s_k, u), S_i - \{s_k\}\rangle$ and $\langle\text{OEFFSTEP}(F'_i, \psi(s_k), u'), S'_j - \{\psi(s_k)\}\rangle$ where $u \in F_i$ and u permutes to u'. For f in F_i and g in F'_i such that $f \longleftrightarrow_{S_i,S'_i} g$, $f(s_k) = u(s_k)$ iff $g(\psi(s_k)) = u'(\psi(s_k))$. Thus:

$$\text{OEFFSTEP}(F_i, s_k, u) \longleftrightarrow_{S_i-\{s_k\},S'_i-\{\psi(s_k)\}} \text{OEFFSTEP}(F'_i, \psi(s_k), u').$$

Additionally, $\forall s \in S_i - \{s_k\}, \psi(s) \in S'_i - \{\psi(s_k)\}$.

Thus by induction, the two trees are isomorphic. Consequently, their tree depths are the same. □

THEOREM 2. *Two subtrees* $\langle F_i, S_u, s_1\rangle$ *and* $\langle F_i, S_u, s_2\rangle$ *have the same tree depth if there exists a one-one mapping of* $\longleftrightarrow_{S_u-\{s_1\},S_u-\{s_2\}}$ *from every set present in the multi-set* $\{\text{OEFFSTEP}(F_i, s_1, u)|u \in F\}$ *to another set in the multi-set* $\{\text{OEFFSTEP}(F_i, s_2, u)|u \in F\}$.

PROOF. Tree depth of subtree from a s-set vertex is calculated by averaging the depth of child f-set vertices (see Figure 1) and adding 1. Because of the mapping, for every subtree depth that we average for s_1, there is a subtree from s_2 with the same depth because of Lemma 1. Thus the averaged values are identical. Consequently, the tree depth from branches selected by s_1 and s_2 are the same. □

4. ANALYZING GENERALIZED LEADING ONES

Now we can use these two techniques on Generalized Leading Ones in an attempt to simplify and analyze it.

4.1 Generalized Leading Ones

We consider a particular generalization of the pseudo-Boolean LEADINGONES problem class (abbreviated here as GLO). For a space of binary sequences of length n, GLO can be defined as follows.

$$\text{GLO}_{\hat{x}}(x) = \sum_{i=1}^{n}\prod_{j=1}^{i}(x_i \equiv \hat{x}_i) \tag{1}$$

where $\hat{x}$ is a sequence of length n hidden in the fitness function. As there are 2^n unique strings in the search space, with each of them as a unique hidden sequence, there are 2^n unique fitness functions in GLO each of them having a unique solution. Put simply, a fitness function in GLO returns the length of the longest prefix common between the sample point and the hidden sequence.

The fitness function for GLO is characterized by large plateaus that make it hard for a randomized search algorithm to optimize locally. These large plateaus occur because changes outside the shared prefix do not influence the fitness value.

Even though some simple EAs can only achieve $\Theta(n^2)$ over GLO, there exists an obvious algorithm that can achieve exactly n: simply flip bits one at a time. It is given as Algorithm 3, where f is the fitness function and on termination, the solution lies in x.

Algorithm 3 Solving GLO one bit at a time

```
binary[N] x ← [0, ..., 0]
for i = 0 to N − 1 do
  if f(x) = i then
    x[i] ← 1
  end if
end for
```

4.2 Building the model

Let $F' = \text{OEFFSTEP}(F, s_k, u)$. For doing state-reduction, using Theorem 1 and the defintion of GLO, we make this observation about OEFF on GLO.

OBSERVATION 2. *$\forall s_i \in S$ such that $u(s_i) \leq u(s_k)$,* $\text{OEFFSTEP}(F', s_i, u) = F'$

One can come to the same conclusion from observing the fitness table of GLO. From the definition of GLO, we know that this is because the fitness function in GLO returns the length of the longest shared prefix common to both the sample point and the solution of the fitness function. As the fitness functions OEFF has not eliminated must also share this prefix, a sample with a shorter prefix cannot eliminate them.

In GLO, as there are exactly $n+1$ unique fitness values being returned by every function, from Observation 2 we can conclude that for a given unknown, a maximum of $n+1$ different elimination sets can be formed by OEFF. But different unknowns may form different elimination sets.

Consider a set F_i^s containing fitness functions sharing a prefix of length i where s is the sample that produced it. As explained later in Equation 3, $u(s) = i-1$. Let us define F_i^s as follows.

$$F_i^s = \left\{ f \;\middle|\; \begin{array}{l} f \in F, \\ f[1\ldots i] = u[1\ldots i] = s[1\ldots i-1]\bar{s}[i] \end{array} \right\} \tag{2}$$

where $u[i\ldots j]$ is the subsequence of the solution of u (as u has a unique solution), from index $i > 0$ up to and including j. The set F_i^s for $0 \leq i \leq n$ and $s \in \{0,1\}^n$ form all the possible sets of un-eliminated fitness functions.

There is no transition from any $F_i^{s_i}$ to any $F_j^{s_j}$ for $j < i$ as $F_j^{s_j} \supset F_i^{s_i}$ if $s_i[1\ldots j-1] = s_j[1\ldots j-1]$ and $F_j^{s_j} \cap F_i^{s_i} = \emptyset$ otherwise. There is a transition for $j > i$ iff $u(s_j) = j-1$. This can be written as follows,

$$\text{OEFFSTEP}(F_i^b, s, u) = \begin{cases} F_i^b & u(s) < i \\ F_{u(s)+1}^s & u(s) \geq i \end{cases} \tag{3}$$

This reduction in states results in a new tree given in Figure 2. Let us examine how optimal sample selection behaves in this tree. For an arbitrary state $F_i^{b_i}$, we will attempt to identify the set of optimal samples which we will prove symmetrical. Consider a sample s such that $s[0\ldots i] \neq b_i[0\ldots i-1]\bar{b_i}[i]$. As $u(s) < i$, the state will remain F_i. Let,

$$S_i^{b_i} = \left\{ s \;\middle|\; \begin{array}{l} s \in \{0,1\}^n, \\ s[0\ldots i] = b_i[0\ldots i-1]\bar{b_i}[i] \end{array} \right\} \tag{4}$$

Consider any two samples $s_1, s_2 \in S_i^{b_i}$. For an arbitrary $u_1 \in F_i^{b_i}$, let $d_1 = u_1(s_1) + 1$.

$$\text{OEFFSTEP}(F_i^{b_i}, s_1, u_1) = F_{d_1}^{s_1} \tag{5}$$

We know there exists $u_1' \in F_i^{b_i}$ such that

$$u_1'(s_2) = u_1(s_1) \tag{6}$$

In fact,

OBSERVATION 3. $\forall u_i, \exists! u_i' \; s.t. \; u_i(s_1) = u_i'(s_2)$

Additionally,

$$\text{OEFFSTEP}(F_i, s_2, u_1') = F_{d_1}^{s_2} \tag{7}$$

THEOREM 3. $F_{d_1}^{s_1} \longleftrightarrow_{S_{d_1}^{s_1}, S_{d_1}^{s_2}} F_{d_1}^{s_2}$.

PROOF. Consider the map M where $\forall y \in \{0,1\}^{n-d_1}$ we map $s_1[1\ldots d_1-1]\bar{s_1}[d_1]y \in S_{d_1}^{s_1}$ to $s_2[1\ldots d_1-1]\bar{s_2}[d_1]y \in S_{d_1}^{s_2}$.

Let $f \in F_{d_1}^{s_1}$. Choose $f' \in F_{d_1}^{s_2}$ such that $f[d_1\ldots n] = f'[d_1\ldots n]$. Let $s_3 \in S_{d_1}^{s_1}$. It is clear that $f(s_3) = f'(M(s_3))$.

In other words, f permutes to f'. The theorem follows from Definition 1. □

Additionally, from $F_{d_1}^{s_1}$, we will not choose a sample point outside $S_{d_1}^{s_1}$. Similarly for $F_{d_2}^{s_2}$. We also know from Observation 3 that the number of $F_{d_1}^{s_1}$ formed by u_i are the same as the number of $F_{d_1}^{s_2}$ formed by u'_i. Thus, from Theorem 2, it follows that all the samples in S_i^b are symmetrical and have the same tree depth.

Thus the best sample selection strategy from F_i would be to randomly pick a sample from $S_i^{b_i}$.

We cannot benefit from Theorem 1 to reduce states further for OEFF since, due to the property of GLO, we already ignore all prior samples.

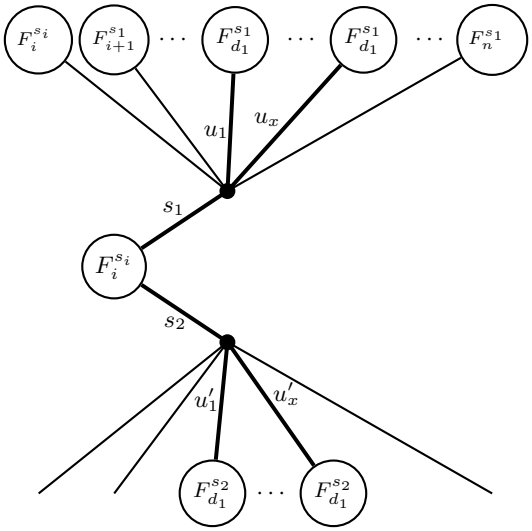

Figure 2: Game tree of OEFF for GLO after state-reduction. Two levels from an arbitrary node $F_i^{S_i}$ is shown.

4.3 Analysis

We can now analyze the behavior of this simplified model and attempt to calculate the black-box performance of GLO.

From an arbitrary state $F_i^{b_i}$, we randomly pick a sample from $S_i^{b_i}$. This sample shares a prefix of length i with the solution. The probability of the $i + j^{\text{th}}$ bit being the first wrong bit is $\frac{1}{2^j}$ for $j \geq 1$. Thus the probability of getting a new fitness $i + j - 1$ is also $\frac{1}{2^j}$. Equation 3 tells us that this is also the probability of transitioning to state F_{i+j}. Using these probabilities to calculate the expectation of j given m higher states,

$$\begin{aligned} E_m(j) &= \sum_{i=1}^{m} \frac{i}{2^i} \\ &= 2 - \frac{m+2}{2^m} \end{aligned}$$

For large n, m can also be expected to be large.

$$\lim_{m \to \infty} E_m(j) = 2$$

Thus for large n, the expected number of samples OEFF with optimal sampling will take to solve Generalized Leading Ones is $\frac{n}{2}$. As the number of samples is a random variable that is a sum of as many independent random variables with finite variance, by law of large numbers, the variance on the black-box complexity approaches 0 for large n. We conclude that this is the exact black-box complexity of Generalized Leading Ones.

An algorithm modeling the behavior of OEFF with optimal sampling can achieve this bound with linear space and time complexity. This is given as Algorithm 4. The function RANDOM returns a random binary sequence of length n.

Algorithm 4 OPTIMALGLO(u)

```
binary[n] s ← RANDOM(n)
f_best ← u(s)
while f_best < n do
  s[f_best + 1] ← s̄[f_best + 1]
  if f_best = n − 1 then
    break
  end if
  s[f_best + 2 ... n] ← RANDOM(n − f_best − 1)
  f_best ← u(s)
end while
return s
```

5. DISCUSSION

We have presented an optimal sampling strategy for the OEFF algorithm. With the help of this sampling strategy, OEFF is a general search method that is optimal over all problem classes given complete domain information. We argue that by modeling OEFF and optimal sampling for a specific problem class, we can design algorithms that are equivalent in behavior, yet avoids the space and time penalties associated with managing domain knowledge externally. We presented a few techniques that can help with the process of modeling. These tools were used to model the behavior of optimal sample selection strategy and OEFF on a well known problem class called Generalized Leading Ones. This model results in a linear expected number of samples required to solve an unknown fitness function from GLO. This confirms the black-box complexity of GLO.

Our bound for the expected first-hitting time for Algorithm 4 is consistent with the general black-box complexity shown in [4]. In fact, the sampling strategy they consider for establishing these bounds is equivalent to Algorithm 4, and it's clear that when one considers success probabilities for our algorithm, the bounds from [4] also hold — it is lower bounded by $n/2 - o(n)$ and upper bounded by $n/2 + o(n)$. While their analysis concerns an abstract process for the purposes of establishing unrestricted black-box complexity bounds, we present a specific and simple algorithm that attains these bounds. Moreover, because we have shown that this is equivalent to OEFF using an optimal sampling strategy, we know that this simple algorithm embeds *all* relevant domain knowledge for this problem class.

Of course, one might have translated the proof used in [4] into an algorithm; however, this is not usually feasible as rarely do optimal algorithms result from deriving the black-box complexity of a domain. We feel the process of modeling OEFF and optimal sampling may lead to promising methods for designing a practical optimal algorithm for a given domain, when one exists.

Another advantage of using OEFF and optimal sampling for deriving black-box complexity is that it provides a single approach for any problem class. The techniques presented in Section 3 are applicable while modeling any problem class. This may simplify the derivation of black-box complexity for some problem classes.

A definite prerequisite of OEFF and any derived algorithm is complete domain knowledge. If domain knowledge is not complete, optimality and/or accuracy of OEFF cannot be guaranteed. However, we are frequently interested in situations where domain knowledge is incomplete. If the domain knowledge that we have can be represented as a probability distribution over all possible problem classes, then, the optimal algorithm may be a combination of OEFF applied to individual problem classes for all problem classes with non-zero probability.

OEFF reminds us that the process of designing an algorithm requires gathering the domain knowledge that exists as objective functions and converting it to search heuristics implemented in an algorithm. The efficiency of a manual algorithm design process can be improved, if it is analyzed with this requirement in mind.

A way to interpret the table of fitness values that OEFF uses during search is as a large number of parameters. This makes OEFF simply a heavily or even the *most* parameterized algorithm. An important lesson is that domain knowledge does not have to be part of the algorithm substantially as search heuristics. It is possible to design algorithms that are more heavily parameterized. From OEFF, we know that such algorithms can gain performance over larger number of domains at the cost of execution time between samples.

Heavily parameterized algorithm may make runtime gathering of domain knowledge easier. Though not a new concept, it has not been applied to heavily parameterized algorithms.

As presented here, the transition from OEFF to the algorithm that models it is made manually. However, it is conceivable that for a restricted set of problem classes, an automated process can study the formal description of the problem class and derive a non-trivial algorithm that is equivalent to OEFF and optimal sampling. This is an interesting possibility that hints at automated design of targeted algorithms for specific problem classes.

In conclusion, we feel OEFF is a very informative algorithm giving us new ideas about performance prerequisites and runtime domain knowledge gathering. OEFF defines a new dimension of algorithms that are heavily parameterized and informs us about the trade-offs in this dimension.

Additionally OEFF and optimal sampling is a promising tool that among other things, can be used for deriving black box complexity of problem classes and designing efficient algorithms that are optimal over specific problem classes. But more attempts at using this tool will be needed before we can be sure of its applicability.

6. REFERENCES

[1] Gautham Anil and R. Paul Wiegand. Black-box search by elimination of fitness functions. In *FOGA '09: Proceedings of the tenth ACM SIGEVO workshop on Foundations of genetic algorithms*, pages 67–78, New York, NY, USA, 2009. ACM.

[2] Benjamin Doerr, Daniel Johannsen, Timo Koetzing, Per Kristian Lehre, Markus Wagner, and Carola Winzen. Faster black-box algorithms through higher arity operators. In *FOGA*, page (unknown), 2011.

[3] Stefan Droste, Thomas Jansen, and Ingo Wegener. Optimization with randomized search heuristics: the (a)nfl theorem, realistic scenarios, and difficult functions. *Theor. Comput. Sci.*, 287:131–144, September 2002.

[4] Stefan Droste, Thomas Jansen, and Ingo Wegener. Upper and lower bounds for randomized search heuristics in black-box optimization. *Theor. Comp. Sys.*, 39(4):525–544, 2006.

[5] Thomas Jansen and Dirk Sudholt. Analysis of an asymmetric mutation operator. *Evol. Comput.*, 18:1–26, March 2010.

[6] Per Kristian Lehre and Carsten Witt. Black-box search by unbiased variation. In *GECCO*, pages 1441–1448, 2010.

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

[8] Mitchell Potter and Kenneth De Jong. A cooperative coevolutionary approach to function optimization. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving from Nature - PPSN III*, volume 866 of *Lecture Notes in Computer Science*, pages 249–257. Springer Berlin / Heidelberg, 1994.

[9] C. Schumacher, M. D. Vose, and L. D. Whitley. The no free lunch and problem description length. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001*, pages 565–570. Morgan Kaufmann, 2001.

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