Unbiased Black Box Search Algorithms

Jonathan E. Rowe
School of Computer Science
University of Birmingham
Birmingham, B15 2TT, UK
J.E.Rowe@cs.bham.ac.uk

Michael D. Vose
Claxton Complex
The University of Tennessee
Knoxville, TN 37996-3450
vose@eecs.utk.edu

ABSTRACT

We formalize the concept of an *unbiased* black box algorithm, which generalises the idea previously introduced by Lehre and Witt. Our formalization of bias relates to the symmetry group of the problem class under consideration, establishing a connection with previous work on No Free Lunch. Our definition is motivated and justified by a series of results, including the outcome that given a biased algorithm, there exists a corresponding unbiased algorithm with the same expected behaviour (over the problem class) and equal or better worst-case performance. For the case of evolutionary algorithms, it is already known how to construct unbiased mutation and crossover operators, and we summarise those results.

Categories and Subject Descriptors

F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

General Terms

Theory, Algorithms

Keywords

Black Box Algorithms, Combinatorial Optimization

1. BACKGROUND

Lehre and Witt consider several restrictions to the classic *black box* optimization scenario, as applied to a search space of binary strings [4]. In particular, they consider algorithms that explore the search space using specialized operators, satisfying the following conditions:

1. The operators are invariant with respect to bit-wise exclusive-or and the permutation of bit positions.

2. Each operator is some function of a single previously sampled element.

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

In this paper we are only concerned with generalizations of the first condition[1]. Lehre and Witt call operators satisfying it *unbiased*, and justify it as "natural" by arguing that the bit operators used in Evolutionary Algorithms are typically unbiased.

We seek to understand this condition more formally, with a view to generalizing it to arbitrary search spaces and to arbitrary problem classes. In so doing, we will demonstrate an intimate connection with the No Free Lunch theorems, and with recent work on characterizing invariance properties of Evolutionary Algorithms [7].

Our results are quite general and apply to all randomized black box search algorithms, and any problem class (defined on a finite search space). This is in contrast to [4] which considers only algorithms of a certain form, and for a particular search space.

2. BLACK BOX ALGORITHMS

We begin by summarizing the basic definitions of Black Box search algorithms, following notation used in [6].

Let $f : \mathcal{X} \to \mathcal{Y}$ be a function between finite sets, and let y_i denote $f(x_i)$. The domain $\mathcal{X}$ and co-domain $\mathcal{Y}$ are fixed for the following discussion, but f may vary.

Definition: A *trace* T *corresponding to* f is a sequence $\langle (x_0, y_0), \ldots \rangle$ of pairs from $\mathcal{X} \times \mathcal{Y}$ where the x components are unique; T is a *trace* if it is a trace corresponding to some f. The following notation will be used,

T^*	=	$\{(x_0, y_0), \ldots\}$	set of pairs of components
T_x	=	$\langle x_0, \ldots \rangle$	sequence of x components
T_y	=	$\langle y_0, \ldots \rangle$	sequence of y components

In particular, $T^* \subseteq f$. The *performance sequence associated with trace* T is T_y; a *performance sequence* is a performance sequence associated with some trace. A function mapping performance sequences to $\mathbb{R}$ is a *performance measure*.

Definition: Trace T corresponding to f is *total* if $T^* = f$. A *partial* trace is one which is not total. The set of all partial traces corresponding to function f is denoted by $\mathcal{T}(f)$, and $\mathcal{T}$ is defined by

$$\mathcal{T} = \bigcup_f \mathcal{T}(f)$$

Definition: A *search operator* is a function $g : \mathcal{T} \to \mathcal{X}$ which maps a partial trace T to some element not occurring

[1]Those readers interested in restrictions on operator *arity* might like to see [1] for further analysis.

in T_x.

Definition: A *deterministic non-repeating Black Box search algorithm* $\mathcal{A}$ corresponds to a search operator g, and will be referred to simply as an *algorithm*. Algorithm $\mathcal{A}$ applied to function f is denoted by $\mathcal{A}_f$, and maps traces to traces

$$\mathcal{A}_f(T) \;=\; \begin{cases} T \,\|\, (g(T), f \circ g(T)) & \text{if } T \in \mathcal{T}(f) \\ T & \text{otherwise} \end{cases}$$

where $\|$ is the concatenation operator.

In procedural terms, algorithm $\mathcal{A}$ runs on function f by beginning with the empty trace $\emptyset$, and repeatedly applying $\mathcal{A}_f$; we denote by $\mathcal{A}(f)$ the total trace produced (by running $\mathcal{A}$ on f to convergence); note that $\mathcal{A}(f)^* = f$. Algorithms $\mathcal{A}$ and $\mathcal{A}'$ are regarded as equal if and only if $\mathcal{A}(f) = \mathcal{A}'(f)$ for all f.

Definition: A *randomized black box search algorithm* is identified with a probability vector μ having components indexed by algorithms, and will be referred to simply as a *randomized algorithm*. The total trace $\mu(f)$ resulting from applying μ to f is $\mathcal{A}(f)$ with probability $\mu_{\mathcal{A}}$.

In procedural terms, randomized algorithm μ runs on f by choosing $\mathcal{A}$ with probability $\mu_{\mathcal{A}}$ and then applying algorithm $\mathcal{A}$ to f. Note that the collection of randomized algorithms contains the set of (deterministic) algorithms.

Randomized algorithms μ and μ' are *equivalent*, denoted by $\mu \equiv \mu'$, if and only if for all f and all T

$$\text{Prob}\{\mu(f) = T\} = \text{Prob}\{\mu'(f) = T\}$$

Following Schumacher [8], given permutation $\sigma : \mathcal{X} \to \mathcal{X}$ and algorithm $\mathcal{A}$ corresponding to search operator g, the algorithm $\sigma\mathcal{A}$ corresponds to the search operator σg, which is defined to act on any given trace T as follows:

$$\sigma g(T) = \sigma^{-1}(g(\sigma_x(T)))$$

and σ_x maps traces to traces as follows: $\sigma_x(\emptyset) = \emptyset$, and if $T = \langle (x_0, y_0), \ldots, (x_{\gamma-1}, y_{\gamma-1}) \rangle$, then

$$\sigma_x(T) = \langle (\sigma(x_0), y_0), \ldots, (\sigma(x_{\gamma-1}), y_{\gamma-1}) \rangle$$

(i.e., σ_x applies σ to the x components of T). Moreover, given $f : \mathcal{X} \to \mathcal{Y}$, define σf to be $f \circ \sigma^{-1}$. Composition of permutations will be denoted by juxtaposition.

Lemma 1. ([3]) For all algorithms $\mathcal{A}$, and permutations σ, σ',

$$\sigma'(\sigma\mathcal{A}) = (\sigma\sigma')\mathcal{A}$$

Moreover, $\iota\,\mathcal{A} = \mathcal{A}$ where ι is the identity permutation.

Following Duenez-Guzman [3], given randomized algorithm μ, define the randomized algorithm $\sigma\mu$ by

$$(\sigma\mu)_{\mathcal{A}} \;=\; \mu_{\sigma^{-1}\mathcal{A}}$$

In procedural terms, to run $\sigma\mu$ on f amounts to choosing $\mathcal{A}$ with probability $\mu_{\mathcal{A}}$ and then running algorithm $\sigma\mathcal{A}$ on f. A performance measure m is extended to randomized algorithms in the natural way; the performance of μ on f as measured by m is

$$m(\mu, f) \;=\; \sum_{\mathcal{A}} m(\mathcal{A}(f)_y)\,\mu_{\mathcal{A}}$$

Note how m is polymorphic: a performance measure m maps performance sequences to values, whereas the performance of μ on f as measured by m is the corresponding expected value.

Theorem 1. ([3]) Given randomized algorithm μ, function f, performance measure m, and permutation σ,

$$m(\sigma\mu, f) = m(\mu, \sigma f)$$

The expected performance of randomized algorithm μ over a set of functions $\mathcal{F} \subseteq \mathcal{Y}^{\mathcal{X}}$ is

$$\mathcal{E}(\mu, \mathcal{F}) \;=\; \frac{1}{|\mathcal{F}|} \sum_{f \in \mathcal{F}} m(\mu, f)$$

and is referred to as expected average performance (one might argue it would be natural to call it average expected performance, but the average and expectation commute).

Lemma 2. ([3]) $\mathcal{E}(\cdot, \cdot)$ is linear in its first argument.

Theorem 2. ([3]) For all randomized algorithms μ, all permutations $\sigma \in \mathcal{X}!$, and all $\mathcal{F} \subseteq \mathcal{Y}^{\mathcal{X}}$,

$$\mathcal{E}(\sigma\mu, \mathcal{F}) \;=\; \mathcal{E}(\mu, \sigma\mathcal{F})$$

Note that choosing $\mathcal{F}$ to be a singleton set reduces Lemma 2 and Theorem 2 to statements about $m(\mu, f)$.

3. UNBIASED ALGORITHMS

Given a search space $\mathcal{X}$, we are often not concerned with the set $\mathcal{Y}^{\mathcal{X}}$ of all functions from $\mathcal{X}$ to some value set $\mathcal{Y}$, but with a subset of $\mathcal{Y}^{\mathcal{X}}$ corresponding to a particular problem class (e.g. functions corresponding to MAXSAT instances). Moreover, we are often not concerned with the set $\mathcal{X}!$ of all permutations of $\mathcal{X}$. Indeed, a criticism of the No Free Lunch theorem is that it only applies to sets of functions which are permutation closed [8], and sets of functions corresponding to traditional combinatorial problem classes do not typically have that property. This leads us to restrict attention to those permutations of the search space which preserve the problem class.

Definition 1. Given $\mathcal{F} \subseteq \mathcal{Y}^{\mathcal{X}}$, let $G(\mathcal{F})$ denote the set of permutations preserving $\mathcal{F}$,

$$G(\mathcal{F}) = \{\sigma \in \mathcal{X}! \,|\, \sigma f \in \mathcal{F} \text{ for all } f \in \mathcal{F}\}$$

Note that $G(\mathcal{F})$ is a subgroup of $\mathcal{X}!$ (both are closed under composition – which is an associative binary operator – and both are closed under the formation of inverses), and

$$\begin{aligned} \sigma G(\mathcal{F}) \;&=\; G(\mathcal{F}) \;=\; G(\mathcal{F})\sigma \\ \sigma\mathcal{F} \;&=\; \mathcal{F} \end{aligned}$$

for all $\sigma \in G(\mathcal{F})$.[2]

A central question left open by [4] is: how do we determine the permutations with respect to which an algorithm should be invariant?

Definition 2. A randomized algorithm μ is *unbiased with respect to a problem class* $\mathcal{F} \subseteq \mathcal{Y}^{\mathcal{X}}$ if and only if $\mu \equiv \sigma\mu$ for all $\sigma \in G(\mathcal{F})$.

[2]We follow the usual convention that $\sigma S = \{\sigma s \,|\, s \in S\}$ and $S\sigma = \{s\sigma \,|\, s \in S\}$.

Examples are presented in the next section illustrating how this definition captures and generalizes concepts in [4]. The following theorems justify the definition, and relate this work to the Focused No Free Lunch theorems of [3, 9].

Theorem 3. Let μ be any randomized algorithm, and let $\mathcal{F} \subseteq \mathcal{Y}^{\mathcal{X}}$ be any problem class. For all $\sigma \in G(\mathcal{F})$,

$$\mathcal{E}(\sigma\mu, \mathcal{F}) \;=\; \mathcal{E}(\mu, \mathcal{F})$$

regardless of the choice of performance measure.

Proof. Since $\sigma \in G(\mathcal{F}) \Longrightarrow \sigma\mathcal{F} = \mathcal{F}$, Theorem 2 yields

$$\mathcal{E}(\sigma\mu, \mathcal{F}) \;=\; \mathcal{E}(\mu, \sigma\mathcal{F}) \;=\; \mathcal{E}(\mu, \mathcal{F})$$

□

When μ is unbiased, the previous result is a triviality, since $\mu \equiv \sigma\mu$. However, when μ is biased, the theorem implies there are non-equivalent algorithms with the same expected average performance. [3, 9].

Definition 3. Given randomized algorithm μ and problem class $\mathcal{F} \subseteq \mathcal{Y}^{\mathcal{X}}$, define the randomized algorithm

$$\mathcal{F}(\mu) \;=\; \frac{1}{|G(\mathcal{F})|} \sum_{\sigma \in G(\mathcal{F})} \sigma\mu$$

Theorem 4. Given randomized algorithm μ and problem class $\mathcal{F} \subseteq \mathcal{Y}^{\mathcal{X}}$,

$$\sigma(\mathcal{F}(\mu)) \;=\; (\sigma\mathcal{F})(\mu) \;=\; \mathcal{F}(\mu)$$

for all $\sigma \in G(\mathcal{F})$. In particular, $\mathcal{F}(\mu)$ is unbiased.

Proof. Lemma 1, together with the fact that $G(\mathcal{F})$ is a group yield

$$\begin{aligned}
(\sigma(\mathcal{F}(\mu)))_{\mathcal{A}} &= \mathcal{F}(\mu)_{\sigma^{-1}\mathcal{A}} \\
&= \frac{1}{|G(\mathcal{F})|} \sum_{\sigma' \in G(\mathcal{F})} (\sigma'\mu)_{\sigma^{-1}\mathcal{A}} \\
&= \frac{1}{|G(\mathcal{F})|} \sum_{\sigma' \in G(\mathcal{F})} \mu_{\sigma'^{-1}(\sigma^{-1}\mathcal{A})} \\
&= \frac{1}{|G(\mathcal{F})|} \sum_{\sigma' \in G(\mathcal{F})} \mu_{(\sigma'\sigma)^{-1}\mathcal{A}} \\
&= \frac{1}{|G(\mathcal{F})|} \sum_{\sigma' \in G(\mathcal{F})} ((\sigma'\sigma)\mu)_{\mathcal{A}} \\
&= \frac{1}{|G(\mathcal{F})|} \sum_{\tau \in G(\mathcal{F})} (\tau\mu)_{\mathcal{A}} \\
&= \mathcal{F}(\mu)_{\mathcal{A}}
\end{aligned}$$

as required. □

Theorem 5. Given randomized algorithm μ and problem class $\mathcal{F} \subseteq \mathcal{Y}^{\mathcal{X}}$,

$$\mathcal{F}(\mathcal{F}(\mu)) \;=\; \mathcal{F}(\mu)$$

Proof. Appealing to Theorem 4,

$$\begin{aligned}
\mathcal{F}(\mathcal{F}(\mu)) &= \frac{1}{|G(\mathcal{F})|} \sum_{\sigma \in G(\mathcal{F})} \sigma\mathcal{F}(\mu) \\
&= \frac{1}{|G(\mathcal{F})|} \sum_{\sigma \in G(\mathcal{F})} \mathcal{F}(\mu) \\
&= \mathcal{F}(\mu)
\end{aligned}$$

as required. □

Note that, so far, we have placed no restrictions on what a performance measure might be. However, it often happens, of course, that we are interested in the task of *optimization* of functions, and we now consider performance measures that may be related to this task.

Definition 4. A performance measure m is *monotone* if either higher performance values are interpreted as superior performance (for example, if we were interested in finding the local optima, and m counted the number encountered before some cutoff time), or else lower values are interpreted as superior performance (for example, if we were interested in finding a global optimum, and m counted the number of function evaluations made before encountering one).

We now show that to any randomized algorithm, there corresponds an unbiased randomized algorithm having equal expected average performance. Moreover, if the performance measure is monotone, the unbiased algorithm cannot have inferior worst-case expected performance.

Theorem 6. Given randomized algorithm μ and problem class $\mathcal{F} \subseteq \mathcal{Y}^{\mathcal{X}}$, the unbiased randomized algorithm $\mathcal{F}(\mu)$ has the same expected average performance as μ. Assuming the performance measure is monotone: $\mathcal{F}(\mu)$ cannot have inferior worst-case expected performance; if the expected performance of μ varies over the orbits of $\mathcal{F}$ under $G(\mathcal{F})$,[3] then $\mathcal{F}(\mu)$ has superior worst-case expected performance.

Proof. Theorem 4 shows $\mathcal{F}(\mu)$ is unbiased. Appealing to Lemma 2 and Theorem 3,

$$\begin{aligned}
\mathcal{E}(\mathcal{F}(\mu), \mathcal{F}) &= \frac{1}{|G(\mathcal{F})|} \sum_{\sigma \in G(\mathcal{F})} \mathcal{E}(\sigma\mu, \mathcal{F}) \\
&= \frac{1}{|G(\mathcal{F})|} \sum_{\sigma \in G(\mathcal{F})} \mathcal{E}(\mu, \mathcal{F}) \\
&= \mathcal{E}(\mu, \mathcal{F})
\end{aligned}$$

To see that the worst-case expected performance cannot be inferior, suppose μ exhibits worst (maximal) performance value at $f^* \in \mathcal{F}$. Appealing to Lemma 2 (see the comment beneath Theorem 2), for any $f \in \mathcal{F}$,

$$\begin{aligned}
m(\mathcal{F}(\mu), f) &= \frac{1}{|G(\mathcal{F})|} \sum_{\sigma \in G(\mathcal{F})} m(\sigma\mu, f) \\
&= \frac{1}{|G(\mathcal{F})|} \sum_{\sigma \in G(\mathcal{F})} m(\mu, \sigma f) \\
&\leq \frac{1}{|G(\mathcal{F})|} \sum_{\sigma \in G(\mathcal{F})} m(\mu, f^*) \\
&= m(\mu, f^*)
\end{aligned}$$

If a minimal performance value means worst performance, then the inequality above reverses, so the same conclusion ($\mathcal{F}(\mu)$ cannot have inferior worst-case expected performance) is obtained. When the expected performance of μ varies over the orbits of $\mathcal{F}$, the worst-case expected performance of $\mathcal{F}(\mu)$ is superior since the inequality is strict. □

Recall that the *black box complexity* of a problem class $\mathcal{F}$ is the minimum worst-case expected optimization time for

[3]The orbit of f is $\{\sigma f \,|\, \sigma \in G(\mathcal{F})\}$, and the orbits of $\mathcal{F}$ are the orbits of $f \in \mathcal{F}$.

functions in $\mathcal{F}$ over all black box algorithms [2]. Consequently, we have

Corollary 1. The black box complexity of a problem class is the same as the unbiased black box complexity of that class.

4. EXAMPLES

4.1 Random search

We now consider random search from the perspective of the formalism developed in sections 2 and 3. Intuitively, each step of random search is conducted by ignoring the partial trace describing how search has so far progressed, and selecting a new random point of $\mathcal{X}$ to visit. By recording the sequence of points visited, one could produce a trace T describing how random search happened to explore the search space. Note that a *deterministic algorithm* — which had trace T hard-coded into its search operator — could explore $\mathcal{X}$ by visiting the same points in the same order. It follows that random search is behaviorally equivalent to randomly choosing and using some deterministic algorithm.

Let E be the "enumeration algorithm" corresponding to the search operator

$$g(T) \;=\; x_{|T|}$$

where $\mathcal{X} = \{x_0, \ldots, x_n\}$ and $|\,T\,|$ is the number of elements in sequence T. It follows that

$$E(f) \;=\; \langle (x_0, f(x_0)), \ldots, (x_n, f(x_n)) \rangle$$

Appealing to the duality theorem (see [6] for example),

$$\begin{aligned}(\sigma^{-1}E)(f) &= \sigma_x(E(\sigma^{-1}f)) \\ &= \sigma_x(E(f \circ \sigma)) \\ &= \sigma_x\langle (x_0, f(\sigma(x_0))), \ldots, (x_n, f(\sigma(x_n))) \rangle \\ &= \langle (\sigma(x_0), f(\sigma(x_0))), \ldots, (\sigma(x_n), f(\sigma(x_n))) \rangle\end{aligned}$$

If we were interested in *uniform random search*, then it is natural to choose σ uniformly (so every enumeration of $\mathcal{X}$ is equally likely); the corresponding randomized algorithm u is defined by

$$u_{\mathcal{A}} \;=\; \frac{1}{|\mathcal{X}!|} \sum_{\sigma \in \mathcal{X}!} [\mathcal{A} = \sigma^{-1}E]$$

where [*expression*] is 1 if *expression* is true, and 0 otherwise. Lemma 1 implies the map $\mathcal{A} \mapsto \sigma\mathcal{A}$ is bijective (its inverse is $\mathcal{A} \mapsto \sigma^{-1}\mathcal{A}$), hence

$$\begin{aligned}\sum_{\mathcal{A}} u_{\mathcal{A}} &= \frac{1}{|\mathcal{X}!|} \sum_{\sigma \in \mathcal{X}!} \sum_{\mathcal{A}} [\sigma\mathcal{A} = E] \\ &= \frac{1}{|\mathcal{X}!|} \sum_{\sigma \in \mathcal{X}!} 1 \\ &= 1\end{aligned}$$

Lemma 3. Given algorithm $\mathcal{A}$, function f, and trace T,

$$\sum_{\sigma \in \mathcal{X}!} [(\sigma\mathcal{A})(f) = T] \;=\; [T^* = f]$$

Proof. If $T^* \neq f$, then $(\sigma\mathcal{A})(f)^* = f \;\neq\; T^*$ and both sides above are zero. Therefore assume $T^* = f$. The proof is completed by showing there exists a unique σ for which $(\sigma\mathcal{A})(f) = T$. Let T^k be the restriction of T to $\{0, \ldots, k\}$.[4] We induct on k to show σ exists such that $(\sigma\mathcal{A})_f^{k+1}(\emptyset) = T^k$. Let $\mathcal{A}$ correspond to search operator g.
Base case $k = 0$:

$$\begin{aligned}(\sigma\mathcal{A})_f(\emptyset) &= \langle (\sigma^{-1}(g(\emptyset)), f(\sigma^{-1}(g(\emptyset)))) \rangle \\ T^0 &= \langle (x_0, y_0) \rangle\end{aligned}$$

Choose σ such that $\sigma(x_0) = g(\emptyset)$.
Inductive case $(\sigma\mathcal{A})_f^k(\emptyset) = T^{k-1}$:

$$\begin{aligned}&(\sigma\mathcal{A})_f((\sigma\mathcal{A})_f^k(\emptyset)) \\ =\;& T^{k-1} \| \, (\sigma^{-1}(g(\sigma_x(T^{k-1}))), f(g(\sigma_x(T^{k-1})))) \\ &T^k \\ =\;& T^{k-1} \| \, (x_k, y_k)\end{aligned}$$

Choose σ such that $\sigma(x_k) = g(\sigma_x(T^{k-1}))$; there can be no conflicts with previous constraints on the choice of σ, since $g(\sigma_x(T^{k-1})) \in \mathcal{X} \setminus \{\sigma(x_0), \ldots, \sigma(x_{k-1})\}$. It follows that there is a unique σ such that $(\sigma\mathcal{A})(f) = T$; all choices in the inductive construction are forced, and the construction defines σ for all $x \in \mathcal{X}$. □

Theorem 7. If – as in the No Free Lunch scenario – the problem class $\mathcal{F}$ is permutation closed ($G(\mathcal{F}) = \mathcal{X}!$), then

$$\text{Prob}\{\mathcal{F}(\mu)(f) = T\} \;=\; \frac{[T^* = f]}{|\mathcal{X}|!}$$

for every randomized algorithm μ.

Proof. Appealing to Lemma 3,

$$\begin{aligned}&\text{Prob}\{\mathcal{F}(\mu)(f) = T\} \\ =\;& \sum_{\mathcal{A}} \mathcal{F}(\mu)_{\mathcal{A}} [\mathcal{A}(f) = T] \\ =\;& \sum_{\mathcal{A}} \frac{1}{|\mathcal{X}!|} \sum_{\sigma \in \mathcal{X}!} (\sigma\mu)_{\mathcal{A}} [\mathcal{A}(f) = T] \\ =\;& \frac{1}{|\mathcal{X}!|} \sum_{\sigma \in \mathcal{X}!} \sum_{\mathcal{A}} \mu_{\sigma^{-1}\mathcal{A}} \, [\mathcal{A}(f) = T] \\ =\;& \frac{1}{|\mathcal{X}!|} \sum_{\sigma \in \mathcal{X}!} \sum_{\mathcal{A}} \mu_{\mathcal{A}} \, [(\sigma\mathcal{A})(f) = T] \\ =\;& \frac{1}{|\mathcal{X}!|} \sum_{\mathcal{A}} \mu_{\mathcal{A}} \sum_{\sigma \in \mathcal{X}!} [(\sigma\mathcal{A})(f) = T] \\ =\;& \frac{1}{|\mathcal{X}!|} \sum_{\mathcal{A}} \mu_{\mathcal{A}} [T^* = f] \\ =\;& \frac{[T^* = f]}{|\mathcal{X}|!}\end{aligned}$$

□

Theorem 8. If – as in the No Free Lunch scenario – the problem class $\mathcal{F}$ is permutation closed ($G(\mathcal{F}) = \mathcal{X}!$), then

$$\mathcal{F}(\mu) \equiv u$$

for every randomized algorithm μ.

Proof. Let e be the randomized algorithm defined by

$$e_{\mathcal{A}} \;=\; [\mathcal{A} = E]$$

[4] A sequence is a function mapping i (from some index set) to the ith element of the sequence.

Note that

$$\begin{aligned}\mathcal{F}(e)_{\mathcal{A}} &= \frac{1}{|\mathcal{X}!|}\sum_{\sigma\in\mathcal{X}!}(\sigma e)_{\mathcal{A}} \\ &= \frac{1}{|\mathcal{X}!|}\sum_{\sigma\in\mathcal{X}!} e_{\sigma^{-1}\mathcal{A}} \\ &= \frac{1}{|\mathcal{X}!|}\sum_{\sigma\in\mathcal{X}!}[\sigma^{-1}\mathcal{A}=E] \\ &= u_{\mathcal{A}}\end{aligned}$$

Hence $u = \mathcal{F}(e)$, and by Theorem 7,

$$\text{Prob}\{u(f)=T\} = \frac{[T^*=f]}{|\mathcal{X}!|} = \text{Prob}\{\mathcal{F}(\mu)(f)=T\}$$

for every randomized algorithm μ. □

Theorems 6 and 8 not only show any algorithm has the same expected average performance as random search (over a permutation-closed problem class), but also that random search has the best worst-case expected performance. It should be appreciated that the extent to which the results in this section are intuitive is one measure of success in our being able to formally demonstrate the appropriateness of our formalism.

4.2 MAXSAT

Let the search space be binary strings of length n and suppose a binary string represents a truth-assignment for a MAXSAT problem on n variables (the ith bit is the value of x_i). Given a MAXSAT instance with objective function f, we are interested in permutations σ such that $f\circ\sigma$ is also a MAXSAT objective function.

One way to obtain such a permutation is to exclusive-or the bits with a fixed binary string. This will have the effect of negating certain literals in the MAXSAT instance. For example, suppose the MAXSAT instance is

$$(x_1, \neg x_3, x_5) \wedge (\neg x_1, x_2, x_4) \wedge (x_2, x_3, \neg x_5)$$

and f is the corresponding objective function which counts how many clauses are satisfied by a given truth assignment. Let σ be the permutation given by applying the mask 11001. Then $f\circ\sigma$ corresponds to the MAXSAT instance

$$(\neg x_1, \neg x_3, \neg x_5) \wedge (x_1, \neg x_2, x_4) \wedge (\neg x_2, x_3, x_5)$$

A second type of permutation preserving this problem class arises from re-ordering the bit positions. Suppose π re-orders the bits as $b_1b_2b_3b_4b_5 \mapsto b_5b_4b_3b_2b_1$. Then $f\circ\pi$ corresponds to the MAXSAT instance

$$(x_5, \neg x_3, x_1) \wedge (\neg x_5, x_4, x_2) \wedge (x_4, x_3, \neg x_1)$$

These types of permutations correspond exactly to those considered by Lehre and Witt to define unbiased operators. Our definition of unbiased with respect to a problem class both captures and generalizes the symmetry concepts they consider in [4].

4.3 Traveling Salesman

Let the search space be the set of all permutations of the labels $\{1, 2, \ldots, n\}$. We consider functions corresponding to traveling salesman problems. Fix a permutation π from the search space. Let σ be the permutation of the search space given by

$$\sigma(x) = \pi \circ x$$

If f is an objective function corresponding to a particular problem instance, then $f\circ\sigma$ corresponds to an instance in which the labels of the original problem have been permuted by π. Unbiased black box algorithms for TSP would have to be invariant with respect to such permutations.

4.4 Subset Sum

The goal of the subset sum problem is to find a non-empty subset of a given a finite set $\{z_1, \ldots, z_n\}$ of integers whose sum is as close as possible to zero. Let the search space again be a set of binary strings representing possible subsets (the ith bit indicates whether the subset contains i). Given the objective function

$$f(x) = \begin{cases} \infty & \text{if } x = 0\cdots0 \\ \sum_i z_i[x_i=1] & \text{otherwise} \end{cases}$$

permutations of the search space given by exclusive-or with a mask (as in the MAXSAT example) will not preserve the problem class, since the value of the zero string will not be infinite. Permutations reordering bit positions are allowed however. We see that for this problem class, the unbiased algorithms are not necessarily invariant with respect to exclusive-or.

4.5 Vertex Cover

Given a graph, the vertex cover problem is to find the smallest set of vertices so that every edge is connected to some vertex in the set. It is easy to see that re-ordering the vertices in the graph gives another problem instance. By our definition, an unbiased algorithm should be invariant with respect to such re-orderings. A genetic algorithm that uses one-point crossover would not be unbiased, because the behavior would depend on the particular vertex ordering.

If we can find a good ordering of vertices, it may happen that one-point crossover is particularly effective. However, incorporating an optimal vertex ordering into the algorithm prevents it from being black box – since knowledge of the problem instance is required. But if we restrict attention to that subset of vertex cover problems for which the vertex ordering is optimal, then our one-point crossover algorithm is black box with respect to that subset. Moreover, since now vertex re-orderings do not preserve the problem class, the algorithm does not need to be invariant with respect to them in order to be unbiased.

5. LINEAR SUBSET PROBLEMS

It would be helpful to have a generic way to deduce the set of permutations which preserve a given search space. We can provide a partial answer for the situation where the problem class has a certain algebraic form, as follows (see chapter 19 of [5] for a similar definition):

Definition 5. A *linear subset problem class* is given by a set C of components and a collection $\mathcal{X}$ of subsets of C, forming the search space. Instances of the problem class are defined by a weight function $w : C \longrightarrow \mathbb{R}$. The objective function is then

$$f(x) = \sum_{i\in x} w(i)$$

Many classical optimisation problems are linear subset problems. We give three examples:

Travelling Salesman Problem The components are all possible edges between cities. Elements of the search space are edge sets forming valid routes. The weight function specifies the problem instance by giving the cost of each edge (with infinity for edges not in the graph).

MAXSAT The components are all possible clauses that can be formed by the literals. An element of the search space is a set of clauses which is satisfied by some truth assignment. The weight function specifies the problem instance by assigning 1 to clauses appearing in the instance and 0 to the remainder.

MAXCUT The components are pairs of distinct vertices. Elements of the search space correspond to subsets of vertices; select components for which one vertex is in the subset and the other is not. The weight function specifies the cost associated with an edge (with zero for edges not in the graph).

Definition 6. Let C and $\mathcal{X}$ be the component set and search space (respectively) of a linear subset problem class. An *automorphism* of the class is a permutation $\pi : C \to C$ for which $\pi(x) \in \mathcal{X}$ for all $x \in X$ (where we apply π element-wise to the members of x).

The set of automorphisms forms a group which acts on $\mathcal{X}$.

Theorem 9. Let $\mathcal{F} \subseteq \mathcal{Y}^{\mathcal{X}}$ correspond to a linear subset problem class. If π is an automorphism of this class, then $\pi \in G(\mathcal{F})$.

PROOF. Let C be the component set. Given any instance corresponding to a weight function $w : C \longrightarrow \mathbb{R}$, we have the corresponding objective function

$$f(x) = \sum_{i \in x} w(i)$$

Note that

$$\begin{aligned}(\pi f)(x) &= f(\pi^{-1}(x)) \\ &= \sum_{i \in \pi^{-1}(x)} w(i) \\ &= \sum_{j \in x} w(\pi^{-1}(j)) \\ &= \sum_{j \in x} (w \circ \pi^{-1})(j)\end{aligned}$$

Hence the objective function πf corresponds to the instance having corresponding weight function $w \circ \pi^{-1}$. □

Some examples:

Travelling Salesman Permutations of the set of possible edges which will preserve tours must correspond to permutations of the cities.

MAXSAT Permutations of clauses preserving the search space include permuting the variables, and negating subsets of literals (and combinations of these). These give us the invariance properties described by Lehre and Witt (and mentioned in the previous section).

MAXCUT The automorphisms include those arising from permuting the labels of the vertices of the graph.

6. EVOLUTIONARY ALGORITHMS

Lehre and Witt [4] constructed unbiased algorithms by using mutation operators which are invariant with respect to exclusive-or and re-orderings of bit positions. It would be helpful – for the purpose of constructing unbiased algorithms – to have a recipe for evolutionary operators guaranteed to produce an unbiased algorithm with respect to any search space and problem class.

Fortunately, much is already known about this problem. In particular, if the group $G(\mathcal{F})$ acts transitively on the search space $\mathcal{X}$,[5] then all mutation and crossover operators invariant with respect to $G(\mathcal{F})$ can be constructed (see [7]).

Mutation is implemented by fixing some element $o \in \mathcal{X}$ and some probability distribution π over $\mathcal{X}$ which has the property: for all $\sigma \in G(\mathcal{F})$ with $\sigma(o) = o$ it is the case that $\pi(x) = \pi(\sigma(x))$ for all $x \in \mathcal{X}$. To mutate an element $x \in X$:

1. Pick $\sigma \in G(\mathcal{F})$ so that $\sigma(o) = x$.
2. Pick $y \in \mathcal{X}$ according to π.
3. Return $\sigma(y)$.

Every mutation operator invariant with respect to $G(\mathcal{F})$ can be implemented in this way, by suitable choice of π.

Crossover is implemented by fixing some element $o \in \mathcal{X}$ and some probability distribution χ over $\mathcal{X}^{\mathcal{X}}$, which has the property: for all $\sigma \in G(\mathcal{F})$ with $\sigma(o) = o$ it is the case that $\chi(h) = \chi(\sigma \circ h \circ \sigma^{-1})$ for all $h \in \mathcal{X}^{\mathcal{X}}$. To cross over two elements $x, y \in \mathcal{X}$:

1. Pick $\sigma \in G(\mathcal{F})$ so that $\sigma(o) = y$.
2. Choose $h \in \mathcal{X}^{\mathcal{X}}$ according to χ.
3. Return $\sigma(h(\sigma^{-1}(x)))$

Every crossover operator invariant with respect to $G(\mathcal{F})$ can be implemented in this way, by suitable choice of χ.

Note that any *selection* operator, since dependent on fitness values and not on the choice of representation of the search space, will always be invariant with respect to any group action on $\mathcal{X}$. Consequently, these results mean that all unbiased evolutionary algorithms can be completely characterised, provided that $G(\mathcal{F})$ acts transitively on $\mathcal{X}$.

Returning to the situation described by Lehre and Witt [4], they require that the mutation operator commute with the group generated by bitwise exclusive-or, and permutation of bit positions acting on the space of binary strings of a given length. Choosing the all zeros string as $o \in \mathcal{X}$, we find that it is permutations of bit positions which preserve o. We therefore require our probability distribution π over $\mathcal{X}$ to have the property that strings containing the same number of ones (or, equivalently, zeros) should have the same probability. Given $x \in \mathcal{X}$, we can take the group element that maps o to x to be

$$\sigma : z \longmapsto z \oplus x$$

We see, then, that any unbiased mutation operator corresponds to first choosing the number of bits to flip (according to some distribution) and then flipping exactly that number of bits (chosen uniformly).

[5] $G(\mathcal{F})$ acts transitively iff $\forall x, y \in \mathcal{X} \,.\, \exists\, \sigma \in G(\mathcal{F}) \,.\, y = \sigma(x)$.

7. CONCLUSIONS

We have given a formal definition of biased and unbiased black box algorithms. The definition of bias is relative to the particular problem class under consideration. There is a surprising link with recent work on focussed No Free Lunch, in that there may be a set of algorithms all having the same expected performance on a problem class that need not be permutation closed. One such algorithm will be unbiased. It will not only have the same expected performance but also equal or better worst-case expected performance than the others. It follows that the black-box complexity of a problem class is the same as the *unbiased* black box complexity of that class. This provides one rigourous justification for the investigation of unbiased algorithms and their performance properties. Moreover, our results apply to any finite problem class on any finite search space. We have also summarized (in the transitive case) the construction of mutation and crossover operators invariant with respect to $G(\mathcal{F})$.

8. ACKNOWLEDGMENTS

Jonathan Rowe would like to thank the organisers and participants of the Dagstuhl Seminar 10361, "Theory of Evolutionary Algorithms".

9. REFERENCES

[1] B. Doerr, D. Johannsen, T. Kötzing, P. K. Lehre, M. Wagner, and C. Winzen. Faster black-box algorithms through higher arity operators. In *FOGA 2011*, 2011.

[2] S. Droste, T. Jansen, K. Tinnefeld, and I. Wegener. A new framework for the valuation of algorithms for black-box optimization. In *Foundations of Genetic Algorithms 7*. Morgan Kaufmann Publishers, 2003.

[3] E. Duenez-Guzman. Biological simulations and biologically inspired adaptive systems. *PhD thesis, The University of Tennesee, Knoxville*, 2009.

[4] P. K. Lehre and C. Witt. Black-box search by unbiased variation. In *GECCO'10*, pages 1441–1448, 2010.

[5] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications; Unabridged edition, 1998.

[6] J. E. Rowe, M. D. Vose, and A. H. Wright. Reinterpreting no free lunch. *Evolutionary Computation*, 17(1):117–129, 2009.

[7] J. E. Rowe, M. D. Vose, and A. H. Wright. Representation invariant genetic operators. *Evolutionary Computation*, 18(4):635–660, 2010.

[8] C. Schumacher. Black box search - framework and methods. *PhD thesis, The University of Tennesee, Knoxville*, 2000.

[9] D. Whitley and J. Rowe. Focused no free lunch theorems. *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, GECCO-2008*, pages 811–818, 2008.