The No Free Lunch and Problem Description Length

C. Schumacher
Department of Computer Science
University of Tennessee, Knoxville
Knoxville, Tennessee 37996 USA
schumach@cs.utk.edu

M. D. Vose and L. D. Whitley
Department of Computer Science
Colorado State University
Fort Collins, Colorado 80523 USA
{vose,whitley}@cs.colostate.edu

Abstract

The No Free Lunch theorem is reviewed and cast within a simple framework for blackbox search. A duality result which relates functions being optimized to algorithms optimizing them is obtained and is used to sharpen the No Free Lunch theorem. Observations are made concerning problem description length within the context provided by the results of this paper. It is seen that No Free Lunch results are independent from whether or not the set of functions (over which a No Free Lunch result holds) is compressible.

1 Introduction

Roughly put, the No Free Lunch theorem formalizes the intuitive idea that all blackbox search algorithms have identical behavior over the set of all possible discrete functions. Thus, on average, no algorithm is better than random enumeration in locating a global optimum. If algorithms are executed any given number of steps, every algorithm finds the same set of best so-far solutions over all functions [9] [5] [1].

One of the criticisms of the No Free Lunch theorem is that it applies to large sets of functions and it is unclear if No Free Lunch applies to small sets or to real world problems of practical interest. A variant form of this criticism is that many practical problem classes have compact descriptions, whereas elements in the set of all functions from a finite domain to a finite codomain do not have (on average) compact descriptions. This criticism has previously been addressed by various researchers [5] [2], where it was observed that a No Free Lunch result holds over classes of functions much smaller than the set of all functions. This paper strengthens those observations, obtaining a sharpened version of the No Free Lunch theorem, and also makes more explicit a type of duality involving functions being optimized and algorithms being used to optimize them. The paper closes with observations regarding the No Free Lunch theorem and problem description length.

2 Search Algorithm Framework

This section sets forth a framework for the analysis of deterministic non-repeating blackbox search algorithms. To streamline exposition, such search algorithms will be referred to simply as algorithms. This framework makes it possible to precisely model all possible algorithms as they apply to all functions of a given finite domain and range.

2.1 Definitions

Let $\mathcal{X}$ and $\mathcal{Y}$ be finite sets, let $f : \mathcal{X} \to \mathcal{Y}$ be a function, and define y_i as $f(x_i)$. Define a *trace* of size m ($m \geq 0$) to be a sequence of pairs

$$T_m \equiv \langle (x_0, y_0), (x_1, y_1), \ldots, (x_{m-1}, y_{m-1}) \rangle$$

Note that a trace is just an ordered sequence of elements from f (regarding f as a set of ordered pairs). At times the subscript of a trace will be omitted to refer to traces of arbitrary size. Let $\mathcal{T}_m$ be the set of all traces of size m, and let $\mathcal{T}$ be the set of all traces. Adopt the following notation:

$$\begin{aligned} T_0 &= \langle\rangle \\ T_m^x &\equiv \langle x_0, x_1, \ldots, x_{m-1} \rangle \\ T_m^y &\equiv \langle y_0, y_1, \ldots, y_{m-1} \rangle \\ T_m[i] &\equiv (x_i, y_i) \\ T_m^x[i] &\equiv x_i \\ T_m^y[i] &\equiv y_i \end{aligned}$$

A concatenation operator $\|$ will be used to extend the size of a trace in the following way:

$$T_m \parallel (x, y) \equiv \langle T_m[0], T_m[1], \ldots, T_m[m-1], (x, y)\rangle$$

Define a *non-repeating trace* T to be a trace with unique x components, i.e. $T^x[i] = T^x[j] \Rightarrow i = j$.[1] A *complete trace* T is defined to be a trace that covers the domain, i.e. for all $z \in \mathcal{X}$ there exists an i such that $T^x[i] = z$. Because a trace is a sequence of ordered pairs, a non-repeating trace corresponds to a function; when it is complete, the corresponding function is f.

Consider a "search" operator $g : \mathcal{T} \to \mathcal{X}$ which when given a trace as an argument returns the next point in the search space to be examined. A *deterministic blackbox search algorithm* A corresponds to a search operator g, and takes as arguments a trace T_m and a function $f \in \mathcal{Y}^{\mathcal{X}}$ and returns the trace

$$T_{m+1} = A_f(T_m) = T_m \parallel (g(T_m), f \circ g(T_m))$$

For example, the first two steps of deterministic blackbox search algorithm A would proceed as follows:

$$\begin{aligned} T_1 = A_f(T_0) &= T_0 \parallel (g(T_0), f \circ g(T_0)) \\ T_2 = A_f(T_1) &= T_1 \parallel (g(T_1), f \circ g(T_1)) \end{aligned}$$

Such algorithms therefore operate in discrete steps where each step generates a new pair that is concatenated into the trace. Note that the search operator g is used to generate the x components of the trace, and that function f is used to evaluate the utility of those points; this reflects the separation between "exploration" (choosing the next point in the search space) and "fitness evaluation" (evaluating the utility of that new point). Multiple applications of these algorithms will be abbreviated in the natural way, i.e. $A_f^m(T_0) = T_m$, and in particular, $A_f^0(T_0) = T_0$.

A *non-repeating blackbox search algorithm*—referred to simply as algorithm—is defined to be a blackbox search algorithm whose range contains only non-repeating traces. The largest trace an algorithm could generate is clearly a complete trace which has size $|\mathcal{X}|$.

After m steps, algorithm A and function f will generate trace T_m from initial trace T_0. In this paper algorithms always start from the empty trace T_0, which may seem a limitation. However, algorithms with an arbitrary initial trace size are actually special cases of algorithms that start from the empty trace, as the following illustrates: Consider algorithm A and initial trace T_m. A corresponds to another algorithm A' that given initial trace T_0 will generate T_m after m steps, and will behave exactly as A afterwards. Designating an initial trace is thus simulated by using a slightly modified algorithm that starts at T_0. In other words, algorithms that can set all points in their traces are powerful enough to encompass algorithms that cannot.

Two algorithms A and B will be considered identical if and only if they both generate the same complete trace for all $f \in \mathcal{Y}^{\mathcal{X}}$, i.e.:

$$A_f^{|\mathcal{X}|} = B_f^{|\mathcal{X}|} \text{ for all } f \in \mathcal{Y}^{\mathcal{X}}.$$

2.2 No Free Lunch

Define a *performance vector* of length m to be a sequence of m values from $\mathcal{Y}$. The performance vector associated with trace T_m is T_m^y. A performance vector can thus be said to be *derived* from a trace, and a function and an algorithm together can be said to generate a performance vector from T_0.

The length m trace $A_f^m(T_0)$ generated by algorithm A and function f will be abbreviated by $T_m(A, f)$. Let $V_m(A, f)$ denote the length m performance vector generated by A and f. The size subscripts may be omitted when not needed. Note that the performance vector $V_m(A, f)$ is closely related to $T_m(A, f)$,

$$V_m(A, f) = (T_m(A, f))^y$$

Define an *overall measure* of algorithm A and set of functions F to be a function that maps the set of performance vectors generated by A and F to a real number. An overall measure can be used to compare the overall performance of two algorithms on a set of functions, and if the two algorithms have identical overall measures, it can be said that they perform equally well over F. An example of an overall measure would be to take a performance vector measure M (which maps a performance vector to a real number), apply it to every element in F and then combine the results in some symmetric way, such as the average $\sum_{f \in F} M(V(A, f))/|F|$.

Define an *No Free Lunch result over* F to be a situation where any two algorithms will have equal overall performance with respect to the set of functions F. Four equivalent statements of the No Free Lunch theorem are given below. Where ambiguous, the set of functions involved is $\mathcal{Y}^{\mathcal{X}}$.

NFL1: For any overall measure, each algorithm performs equally well.

The following is the pivotal idea contained in the proof

[1] This paper will follow the convention that free variables are universally quantified.

by Radcliffe and Surry [5], phrased more directly in the language of the current framework.

NFL2: For any two algorithms A and B, and for any function f, there exists a function g such that $V(A,f) = V(B,g)$.

The following is the basis of the No Free Lunch proof given by Schumacher[6].

NFL3: Every algorithm generates precisely the same collection of performance vectors when all functions are considered.

Schumacher has proved in addition that $V(A,f) = V(A,g) \Longrightarrow f = g$, i.e., the collections referred to in NFL3 are actually sets [6]. This fact is a key observation in demonstrating the equivalence of NFL1, NFL2, NFL3, and NFL4.

As defined above, an overall measure is a function of a set of performance vectors. Consider instead a *weighted overall measure* in which a performance vector measure M applied to each performance vector is weighted according to the function that generates it, i.e. $W(f)M(V(A,f))$, and summed over f. A weighted overall measure is *not* generally subject to the No Free Lunch theorem except in the case where the functions are *equally weighted*, i.e. certain functions are not deemed more important than others. The statement below is essentially the No Free Lunch result given in Wolpert and Macready [8].

NFL4: For any equally weighted overall measure, each algorithm will perform equally well.

A corollary of the No Free Lunch theorem is that if an algorithm performs better than average on one set of functions, it must perform worse on the complementary set. This is essentially an argument for specialization: an algorithm will perform well on a small set of functions at the expense of poor performance on the complementary set.

An even stronger consequence which seems not to have been properly appreciated is that *all* algorithms are equally specialized. This contradicts commonly stated beliefs (e.g. [4]) about how there can be *robust* general purpose algorithms, meaning that they perform reasonably well on a broad class of functions at the expense of not performing extremely well on any set of functions. Since every algorithm has precisely the same collection of performance vectors when all functions are considered (NFL3), it follows that *if any algorithm is robust, then every algorithm is, and if some algorithm is not robust, then no algorithm can be!*

3 Sharpening No Free Lunch

Let $f : \mathcal{X} \to \mathcal{Y}$ be a function and let $\sigma : \mathcal{X} \to \mathcal{X}$ be a permutation (i.e. σ is one-to-one and onto). The permutation σf of f is the function $\sigma f : \mathcal{X} \to \mathcal{Y}$ defined by $\sigma f(x) = f(\sigma^{-1}(x))$.

Define a set F of functions to be *closed under permutation* if for every $f \in F$, every permutation of f is also in F.

Let A be an algorithm with search operator g and let σ be a permutation (of $\mathcal{X}$). The permutation σA of A is the algorithm with search operator σg defined by $\sigma g(\phi) = \sigma^{-1}(g(\sigma_x(\phi)))$ where $\sigma_x(\phi)$ operates on the x values of trace ϕ by applying σ to each of them, while leaving the y values untouched.

THEOREM: If

$$T_n(A, \sigma f) = \langle (x_0, y_0), \ldots, (x_{n-1}, y_{n-1}) \rangle$$

then

$$T_n(\sigma A, f) = \langle (\sigma^{-1}(x_0), y_0), \ldots, (\sigma^{-1}(x_{n-1}), y_{n-1}) \rangle$$

Proof: By induction on the length of the traces. The base case is true since all traces of length 0 are the same; $T_0(\sigma A, f) = T_0(A, \sigma f) = \langle\rangle$. Assume the inductive hypothesis (i.e., the equalities in the statement of the theorem). By definition,

$$\begin{aligned} \sigma g(T_n(\sigma A, f)) &= \sigma^{-1} \circ g(\sigma_x(T_n(\sigma A, f)) \\ &= \sigma^{-1} \circ g(T_n(A, \sigma f)) = \sigma^{-1}(x_n) \end{aligned}$$

Moreover, $f(\sigma^{-1}(x_n)) = \sigma f(x_n) = y_n$. Accordingly:

$$\begin{aligned} T_{n+1}(A, \sigma f) &= T_n(A, \sigma f) \mid\mid (x_n, y_n) \\ T_{n+1}(\sigma A, f) &= T_n(\sigma A, f) \mid\mid (\sigma^{-1}(x_n), y_n) \end{aligned}$$

Which completes the proof. □

COROLLARY ("Duality"): $V(\sigma A, f) = V(A, \sigma f)$

This Corollary is true since, by the previous theorem, the y values are the same in both traces. This corollary is striking in the way that it shows a correspondence between a permutation of an algorithm and a permutation of a function. The following Lemma is an easy consequence of NFL2.

LEMMA1: If the set of functions F is closed under permutation, then there is a No Free Lunch result over F.

Proof: Let A and B be arbitrary algorithms. If one can show the sets $S_1 = \{V(A,f) : f \in F\}$ and $S_2 = \{V(B,h) : h \in F\}$, are equal, then any two algorithms will provide the same data for computing

their combined performance measures, and therefore the same result will be obtained. By NFL2, there exists a function h such that $V(A,f) = V(B,h)$. Because these two performance vectors are equal, h must be a permutation of f, and thus $f \in F \Longrightarrow h \in F$. Hence $S_1 \subset S_2$. The reverse containment follows by symmetry. □

The previous lemma was an intermediate result in Radcliffe and Surry's proof of the No Free Lunch theorem [5]. The converse of this lemma is also true.

LEMMA2: If a No Free Lunch result holds over the set of functions F, then F is closed under permutation.

Proof: Assume by way of contradiction that a No Free Lunch result holds over the set F, but that F is not closed under permutation, i.e., the function $f \in F$ has a permutation g which is not in F. Consider an arbitrary algorithm A. Let $M(V(A,f)) = 1$, and let M equal zero for all other performance vectors generated by A. By NFL3 and the paragraph following it, for every algorithm B there exists a function h_B (the subscript on h indicates dependence on B) such that $M(V(B,k)) = 1 \iff k = h_B$. Let the overall measure be the sum $\sum_{k \in F} M(V(B,k))$. Note that this sum is 1 when $B = A$, and since a No Free Lunch result is assumed over F, the sum is 1 for every algorithm B. As f and g are permutations, let $f = \sigma g$. By duality, $V(A,f) = V(A,\sigma g) = V(\sigma A, g)$, and thus $M(V(\sigma A, g)) = 1$. Accordingly, $\sum_{k \in F} M(V(\sigma A, k)) = 0$ (since $M(V(\sigma A, k))$ is non zero only for $k = h_{\sigma A} = g \notin F$), a contradiction. □

Combining the previous lemmas yields the following sharpened version of the No Free Lunch theorem:

NFL: A No Free Lunch result holds over the set of functions F if and only if F is closed under permutation.

4 NFL and Permutation Closure

In this section, some consequences of the previous results are illustrated.

Define the permutation closure $P(F)$ of a set of functions $F \subset \mathcal{Y}^{\mathcal{X}}$ by

$$P(F) = \{\sigma f : f \in F, \text{ and } \sigma \text{ is a permutation (of } \mathcal{X})\}$$

Note that for any sets F, F' of functions (from $\mathcal{Y}^{\mathcal{X}}$),

$$P(F \cup F') = P(F) \cup P(F')$$

By construction, $P(F)$ is closed under permutation and therefore a No Free Lunch result holds over $P(F)$ for any set $F \subset \mathcal{Y}^{\mathcal{X}}$ (and hence over unions of such sets). It bears mentioning that in particular NFL1, NFL2, NFL3, and NFL4 are valid with respect to $P(F)$. Not only do all algorithms display equal behavior over $P(F)$ for some overall measure of performance (NFL1), they also generate exactly the same set of performance vectors (NFL3) and therefore have identical collections of objective function values at every time step.

An equivalence relation $\equiv$ may be defined with respect to permutations. Functions f and g are said to be equivalent, denoted by $f \equiv g$ if and only if there exists a permutation σ (of $\mathcal{X}$) for which $f = \sigma g$. Similarly, algorithms A and B are said to be equivalent, denoted by $A \equiv B$ if and only if there exists a permutation σ (of $\mathcal{X}$) for which $A = \sigma B$.

Let the equivalence class of function f be denoted by $[f]$, and let the equivalence class of algorithm A be denoted by $[A]$. To simplify notation, let $\mathcal{A}$ denote a set of algorithms, let $\mathcal{F}$ denote a set of functions, and define $V(A,\mathcal{F})$ and $V(\mathcal{A},f)$ as follows

$$\begin{aligned} V(A,\mathcal{F}) &= \{V(A,f) : f \in \mathcal{F}\} \\ V(\mathcal{A},f) &= \{V(A,f) : A \in \mathcal{A}\} \end{aligned}$$

Since $[f] = P(\{f\})$, NFL applies; therefore, for any given algorithms A and B,

$$V(A,[f]) = V(B,[f])$$

It follows immediately from the definitions that if F is closed under permutation and $f \in F$ then $[f] \subset F$. Therefore the case above (i.e., $F = [f]$) is the finest level of granularity at which a No Free Lunch result can hold. Moreover, any set F of functions closed under permutation is a disjoint union of equivalence classes, thus No Free Lunch results hold only over unions of equivalence classes.

By definition and duality,

$$\begin{aligned} V([A],f) &= \{V(\sigma A, f) : \sigma \text{ is a permutation}\} \\ &= \{V(A,\sigma f) : \sigma \text{ is a permutation}\} \\ &= V(A,[f]) \end{aligned}$$

Bringing NFL into the picture yields the result that for any given algorithms A and B,

$$V([A],f) = V(A,[f]) = V(B,[f]) = V([B],f)$$

It follows that for any given algorithm A and any given function f, the following are identical:

- The average performance over all algorithms using function f.

- The average performance over an arbitrary equivalence class of algorithms using function f.
- The average performance over all functions in the equivalence class $[f]$ using algorithm A.

Moreover, the phrase "average performance" can be replaced with "set of performance vectors" in the list above. Whereas most No Free Lunch results have been expressed in terms of some measure of performance, all algorithms in fact display exactly the same behavior over any set of functions closed under permutation in the sense that the performance vectors are identical.

5 NFL Equivalence Class Examples

In this section, some extreme examples of permutation closure are presented. These examples not only illustrate applications of NFL, but also set the stage for discussing the notion of problem description length.

For conciseness, a function will be represented by a list of its output values (i.e., as a sequence; the points of the domain are implicitly the indices into the sequence).

The smallest permutation closures correspond to functions that return a single value. For example,

$$f = \langle 0,0,0,0 \rangle \implies [f] = \{f\}$$

Such problems are in some sense uninteresting from a search point of view, since a single evaluation automatically determines the maximum and minimum of the evaluation function.

The smallest sets corresponding to a permutation closure where the evaluation functions display variability are needle-in-a-haystack functions. Such a function f has the same evaluation (call it 0) everywhere except at one point in the domain, where a better evaluation is found (call it 1). Since there is exactly one point in the space with a different evaluation, the size of $[f]$ is $|\mathcal{X}|$. For example

$$\begin{aligned} f &= \langle 0,0,0,1 \rangle \implies \\ [f] &= \{\langle 0,0,0,1 \rangle, \langle 0,0,1,0 \rangle, \langle 0,1,0,0 \rangle, \langle 1,0,0,0 \rangle\} \end{aligned}$$

An interesting class of functions is the set of decision problems which return Boolean values ($\mathcal{Y} = \{0,1\}$). Note that NP-Complete problems are frequently defined as particular decision problems. This class is in one-to-one correspondence with the set of length $N = |\mathcal{X}|$ binary strings, and is therefore equal to its permutation closure. It is moreover a disjoint union of equivalence classes $[f_1], \ldots, [f_N]$ where

$$\begin{aligned} f_i &= \langle \underbrace{1 \ldots 1}_{i \text{ times}}, 0, \ldots, 0 \rangle \\ |[f_i]| &= \binom{N}{i} \end{aligned}$$

As a final example, consider functions that are one-to-one and onto. This class of functions also equals its permutation closure. Without loss of generality, $\mathcal{X} = \mathcal{Y}$ and such functions are permutations. There is a single equivalence class, namely $[I]$ where I is the identity function, and its size is $N!$ (where $N = |\mathcal{X}|$).

6 Problem Description Length

The average description length for functions that are members of a permutation closure is discussed in this section. Although this is only done for select cases, the cases illustrate the extremes in average description length.

Whitley [7] has previously made (a variation on) the observation that given any permutation σ (on $\mathcal{X}$), the permutation closure $[\sigma]$ is the set of all N! permutations, and the average description length for its members is $\Omega(N \ln N)$ bits, where $N = |\mathcal{X}|$.

A more general observation is that given any set F of functions, the average description length of members in $P(F)$ is $\Omega(\ln k)$ bits, where $k = |P(F)|$.

An interesting question is: when is the the average description length over the members of some permutation closure polynomial and when it it exponential? A correct, but somewhat circular answer, is that the average description length is polynomial when $\ln k$ is polynomial. Nevertheless, we can still use this idea to examine average description length for examples which provide bounding cases.

It has already been noted above that the average description length for permutations is $\Omega(N \ln N)$ bits. Note, moreover, that an explicit definition of a permutation (as a sequence, as described in the previous section) would take $O(N \ln N)$ bits; there are N images (positions in the sequence) to define, and each takes $O(\ln N)$ bits (since there are N points in the range). Therefore, on average, the permutation closure $[\sigma]$ contains *incompressible functions*; the average description length of a member is the same order of magnitude as the size of an explicit definition (as a sequence).

At the other extreme is the permutation closure $[f]$ of a needle-in-a-haystack function f (described above).

Explicit definition of a member (of $[f]$) requires $\Omega(N)$ bits, whereas the average description length is $O(\ln N)$ bits. Therefore members of this permutation closure are highly compressible.

These two extreme cases illustrate that No Free Lunch results are independent from whether or not the set of functions (over which a No Free Lunch result holds) is compressible.

7 Conclusions

No Free Lunch theorems in various equivalent forms are reviewed. A duality result is proven and used to obtain a sharpened No Free Lunch theorem, in the sense that both necessary and sufficient conditions are obtained.

It is proven that the permutation closure of a single function is the finest level of granularity at which a No Free Lunch result can hold. The average description length of members of permutation closures is computed (for select cases) and is related to compressibility. It is seen that No Free Lunch results are independent from whether or not the set of functions (over which a No Free Lunch result holds) is compressible.

References

[1] J. Culberson. On the Futility of Blind Search. *Evolutionary Computation*, 6(2):109–127, 1999.

[2] T. English. Practical implications of new results in conversation of optimizer performance. In Schoenauer et al., editor, *Parallel Problem Solving from Nature, 6*, pages 69–78, 2000.

[3] Thomas English. Information is Conserved in Optimization. *IEEE Trans Evolutionary Computation.*

[4] David Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[5] N.J. Radcliffe and P.D. Surry. Fundamental limitations on search algorithms: Evolutionary computing in perspective. In J. van Leeuwen, editor, *Lecture Notes in Computer Science 1000.* Springer-Verlag, 1995.

[6] C. Schumacher. *Fundamental Limitations of Search.* PhD thesis, University of Tennessee, Department of Computer Sciences, Knoxville, TN, 2000.

[7] D. Whitley. Functions as permutations: regarding no free lunch, walsh analysis and summary statistics. In Schoenauer et al., editor, *Parallel Problem Solving from Nature, 6*, pages 169–178, 2000.

[8] David H. Wolpert and William G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, July 1995.

[9] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 4:67–82, 1997.