What can we learn from No Free Lunch?
A First Attempt to Characterize the Concept of a Searchable Function

Submission category: Methodology, Pedagogy, and Philosophy

Steffen Christensen

Computer Science Dept.
Carleton University
Ottawa, ON K1S 5B6
steffen@scs.carleton.ca
(613) 520-2600 x 1857

Franz Oppacher

Computer Science Dept.
Carleton University
Ottawa, ON K1S 5B6
oppacher@scs.carleton.ca
(613) 520-2600 x 3520

Abstract

The No Free Lunch theorem has had considerable impact in the field of optimization research. A terse definition of this theorem is that no algorithm can outperform any other algorithm when performance is amortized over all functions. Once that theorem has been proven, the next logical step is to characterize how effective optimization can be under reasonable restrictions. We operationally define a technique for approaching the question of what makes a function searchable in practice. This technique involves defining a scalar field over the space of all functions that enables one to make decisive claims concerning the performance of an associated algorithm. We then demonstrate the effectiveness of this technique by giving such a field and a corresponding algorithm; the algorithm performs better than random search for small values of this field. We then show that this algorithm will be effective over many, perhaps most functions of interest to optimization researchers. We conclude with a discussion about how such regularities are exploited in many popular optimization algorithms.

1 INTRODUCTION

The No Free Lunch Theorem (NFL) states that no single algorithm outperforms random search (equivalently, systematic linear search) when amortized over all functions (Wolpert and Macready, 1995). This is easy to see and, indeed, unsurprising: if nothing is known about the structure of a domain, then all problems in it are equally likely; and any particular search or learning algorithm can be expected to perform better than chance on some randomly selected problems and worse than chance on others, and on average it can be expected to do no better (and no worse) than random guessing.

The apparently disheartening conclusion of NFL depends on the assumption that nothing is known about the structure of a domain; by the same token, the theorem may be taken to point out the importance of assumptions about non-uniformities in a domain for an understanding of the success of search and learning algorithms.

In this paper we try to specify general and minimal non-uniformities that seem to be realized in nearly all application domains, and that enable certain algorithms to outperform random search. In particular, we believe that virtually all functions that a practitioner would consider searching exhibit the requisite non-uniformity and are thus unaffected by NFL.

In the following, we consider what makes a function searchable, and then give an algorithm that outperforms random search on functions thus characterized.

2 WHAT MAKES A FUNCTION SEARCHABLE?

We all know that so-called "general-purpose" optimization algorithms are at least modestly effective on a wide range of combinatorial optimization problems. In fact, most general-purpose optimization algorithms greatly outperform random search on functions commonly used in testing these algorithms. Indeed, while random and linear search find minima in time linear in the size of the search space, these algorithms are running in time exponential in the input.

Most "general-purpose" optimization techniques rely on some sort of hill climbing at the lowest level. These techniques include golden section search, Brent's method, the downhill simplex method, direction-set methods, conjugate gradient methods, quasi-Newton methods, simulated annealing, the genetic algorithm, evolution strategies, evolutionary programming and various types of

hill climbers themselves (Press, 1992; Fogel, 1995; Mitchell, 1996; Schwefel, 1977). However, NFL guarantees even hill climbers will not outperform random search when amortized over all functions.

If all these methods are exploiting similar general and minimal sorts of regularities in the functions that they successfully search, then perhaps these regularities are encountered more often in functions of interest in the real world than the uniform distribution over which NFL applies would suggest. How can we reconcile our intuition and our experience with the efficacy of different types of search algorithms with the conclusion of NFL?

2.1 EXPERIMENTAL FRAMEWORK

We define a framework for talking about optimization problems in general, adopting the notation given in Macready and Wolpert (1996).

In general optimization problem, we are given a cost function $f : X \rightarrow Y$ to optimize. Without loss of generality, we assume that we are seeking the global minima of f. Since optimization techniques will in general be implemented on digital computers, we can take X and Y to be finite sets. (For instance, Y might be the set of floating-point numbers representable in a computer's native hardware representation, while X may be an ordered set of such floating-point numbers.) These sets are of size $|X|$ and $|Y|$, respectively.

A "population" d_m is defined as an ordered sample of m successive points chosen by a given algorithm from the cost function. $d_m \equiv \{d_m(i)\} \equiv \{d_m^x(i), d_m^y(i)\}$. Note that this definition differs from the concept of population used in the evolutionary computation literature, in that it includes *all* of the points ever evaluated, while in evolutionary computation we normally consider a population to be a working set of points in the domain that may or may not overlap with points selected in previous generations.
Let D_m be the set of all populations of size m, and let $D = \bigcup_m D_m$ be the set of all populations of any size. An algorithm that attempts to find minima of f can be made maximally efficient if it does not revisit any points. (This can be implemented in any standard optimization algorithm by caching the function evaluations, and by looking up subsequent evaluations from the cache.) We can therefore define an algorithm as a mapping $a: d \in D \rightarrow \{x \mid x \notin d^x\}$.

Let $\omega(f)$ be a vector of length $|Y|$ such that each element ω_i is the number of points $x \in X$ such that $f(x) = \omega_i$. That is, $\omega(f)$ is the histogram of the y-values of f. Define $\Omega(f)$ as the vector of the successive nonzero components of $\omega(f)$. $\Omega(f)$ thus corresponds to the ordered non-zero histogram of the distribution of values of f. f_1 and f_2 are then said to be in the same equivalence class iff $\Omega(f_1) = \Omega(f_2)$. That is, two functions are in the same equivalence class if the same number of points in X map to each corresponding point in Y. Macready and Wolpert make several important points about trying to compare the relative hardness of functions taken from different equivalence classes in (Macready and Wolpert, 1996); we refer the reader to their discussion therein for more information.

It is our belief that general-purpose optimization of a particular random function about which nothing is known *a priori* can only hope to be successful if there is a degree of "self-similarity" in the function. By self-similarity, we mean that function values of points that are "near" to a given point are "related" to that point's value. Such self-similarity is, in general, difficult to characterize completely; it seems that a truly accurate definition would have to rely on the notion of Kolmogorov complexity such as "what is the ratio of the size of the Turing machine of minimum length that can reproduce the patterns observed in the function to the size of the function's domain?" However, such a measure would be computationally intractable. The approach we choose in this paper is to define a technique that allows us to specify, investigate and analyze our own operational measure of self-similarity. Thus, if you have in mind a specific kind of regularity or pattern that your optimization technique effectively exploits, you may be able to operationally define that regularity and make concrete statements about the effectiveness and generality of your optimization technique.

We begin by requiring a total order of all points in the domain and in the range. For traditional multidimensional optimization in digital computers, we can simply take the order of the complete binary representation of the points in memory. For fixed-point and floating-point discretizations, this is equivalent to saying that larger values in the first dimension sort higher than lower values, and that relative values in succeeding dimensions break ties.

Without loss of information, we can define a function isomorphism that maps the elements of the domain of our original function f to the integers in $[1..\delta]$, and maps the elements of the range to the integers in $[1..\rho]$. (δ and ρ are used to suggest "size of domain" and "size of range", respectively. Note that δ is the size of the preimage of f, while ρ is the size of the image of f.) The simplest such isomorphism is to index the elements of X and Y, and in our new function, preserve the mapping of the indices of X to the indices of Y. We designate this generated function $r(f)$, and call the set of all such functions R, $R : \mathbb{N}_\delta \rightarrow \mathbb{N}_\rho$. The *successor* graph of such a function r is

the matrix $G(r)$ whose elements g_{ij} are defined by[1]:

$$g_{ij} = \sum_{i=0}^{\delta-1} [r(i+1) = j].$$

Note that $\sum_{j=1}^{\rho} g_{ij}$ = the i-th component of the previously introduced index function $\Omega(f)$. Furthermore, invertible functions have $\rho = \delta$, and therefore all components of $\Omega(f)$ are 1. A random function will, in general, exhibit no relation between $r(i)$ and $r(i+1)$; therefore the corresponding matrix $G(r)$ will reveal few or no patterns to exploit.

2.2 TOWARDS UNDERSTANDING OPTIMIZATION

We now begin a general process to classify the functions taken from a function space such as R with respect to the NFL theorem. As Wolpert and Macready state (1995), algorithms that do better in some regions of R must do correspondingly worse in others. To attempt to characterize which functions are "searchable", we adopt the following procedure: we define a computable scalar field S over R; we then show a systematic relation between the values of this field on functions in R and the performance of some corresponding algorithm A. (Note that this concept may readily be extended to include vector fields over R.)

If we can then show that the performance of this algorithm A will be better in areas of R where $S(r)$ is high, we will have effected a partition of the function space R based on searchability. Thus, we can begin to operationally address the question of how well we *can* do in general-purpose optimization, *given* that NFL holds.

By way of explanation, we here offer a definition of one such field and a corresponding algorithm. The goal of this algorithm, SUBMEDIAN-SEEKER, is to choose points in the search space that lie below the median of r as early in the selection process as possible.

We define the field $M(r)$ as follows:

Let $M(r) = \sum_{i=0}^{\rho/2} \sum_{j=\rho/2+1}^{\rho} g_{ij}$, where g_{ij} is the i, j-th component of $G(r)$. Thus $M(r)$ is the number of function values that lie below the median whose successors' values lie above the median.

We now define a general-purpose optimization algorithm, SUBMEDIAN-SEEKER, which attempts to use information gathered on the points observed thus far to predict future points. In the following, we express the algorithm in terms of a random function f, for generality. Because of the possibility of constructing a function isomorphism between any real-valued function f and the corresponding ranked function r, the algorithm applies equally in both cases. However, the analysis is simplified if we continue to work in the uniform function space, R.

2.3 ALGORITHM

SUBMEDIAN-SEEKER(f):

$s \leftarrow \{0,0\}$ -- *will count the number of sampled points with successors below the median, for the first and second halves of the domain.*

$z \leftarrow \{0,0\}$ -- *will count the number of sampled points with successors above the median*

$i \leftarrow 0$

$j \leftarrow 0$

Until j = *Number of points to sample* do:

  Choose a next point for the algorithm to consider as follows:

    $window \leftarrow [f(x_i) < median(f)]$

    If $s_{window} > z_{window}$ then

      $x_{i+1} \leftarrow x_i + 1$

    Else

      $x_{i+1} \leftarrow$ *Random unchosen point*

  $i \leftarrow i + 1$

  Evaluate $r(x_i)$.

  $j \leftarrow j + 1$.

  If $j \neq$ *Number of points to sample* then

    If $f(x_i + 1)$ has not been evaluated then evaluate $f(x_i + 1)$ and set $j \leftarrow j + 1$.

  Add the $f(x_i)$ vs. $f(x_i + 1)$ information to the graph $\boldsymbol{G}$ as follows:

    If $f(x_i) < med(f)$, then increment $s_{[f(x_i) >= med(f)]}$

    Else increment $z_{[f(x_i) >= med(f)]}$.

2.4 ALGORITHM SUMMARY

This algorithm basically builds up information about the function by sampling. s and z store information on the number of values that map to successor values below and above the median, respectively. This information is used to guide further progress in the search. This algorithm can be expected to perform well if previous performance is an accurate indicator or predictor of future performance. There is, of course, no guarantee that this is the case; however, if many points in the function have values and successor values that happen to co-occur below the median, then SUBMEDIAN-SEEKER will be able to exploit these regularities. We will see later that many functions that we might attempt to search will have this characteristic.

[1]Note that here and elsewhere in this paper, we use Knuth's shorthand notation for predicate functions (Graham, Knuth, and Patashnik, 1994): $[x = y]$ is a terse way of specifying the function $\begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$.

2.5 EXPERIMENTATION ON SUBMEDIAN-SEEKER

We define the performance $\phi(a, r)$ of an algorithm a as the number of points discovered in m samples that are below the median. In Wolpert and Macready's notation, (Wolpert and Macready, 1995):

$$\varphi(a \mid r, m) = \sum_{i=1}^{m} [d_i^y < \rho / 2]$$

We would expect that any differences between the distribution of points selected by SUBMEDIAN-SEEKER and that of a random-search algorithm would be maximized at $m = \delta/2$; after this point, the requirement to sample only previously unselected points would force the algorithm to begin making unfavourable choices. We therefore run the algorithm until it has chosen exactly half the points in the domain - it is expected (and verified by experiment) that this will maximize any performance difference between random search and SUBMEDIAN-SEEKER.

Therefore, in every test in the present work, we ran the algorithm until exactly half of the points in the domain were selected. (It can be inferred from this that we generally chose to investigate function with domains of modest size, unlike those used in serious optimization research!) For simplicity, we worked only over the invertible functions I_n, with $\delta = \rho = n$.

Since SUBMEDIAN-SEEKER is a stochastic algorithm (i.e. the points it chooses are selected at random if no other information is available), it will generate a distribution of outcomes for any given source function r. However, this distribution is fixed for a given r. We are therefore free to probe the distribution of outcomes by rerunning the algorithm as many times as is necessary to ascertain the true distribution as precisely as is required.

There are many choices for how one could evaluate the performance of SUBMEDIAN-SEEKER, particularly since its performance will be contrasted with that of random search, another stochastic algorithm. The measure used herein is to compare the mean performance of SUBMEDIAN-SEEKER with the (known) mean performance of random search using the Student's t-test. In general, the use of a t-test to ascertain the difference in means of two random discrete distributions is not valid, because of the assumptions that the t-test makes of normality and homoscedasticity of the variables (see Neter et al, 1996, pp. 407-408). Therefore the accuracy of this simplification was explicitly tested using a Monte Carlo simulation technique. 10 000 simulation runs of 100 experiments each were performed, and the run data were used to estimate parameters of the observed distributions, such as the mean and standard deviation. T-tests were then performed for each run to attempt to reject the incorrect hypothesis that the population mean was greater than a predetermined threshold. The agreement between the observed confidence levels and those estimated from observed distributions using the Student's t-test were excellent. In addition, we used very large confidence intervals to compensate for this uncertainty.

Note that the only test that SUBMEDIAN-SEEKER performs is to see whether the value of the cost function is above or below the median. (For the moment, we assume that there are no points exactly at the median of r; i.e. δ is even.) Therefore, the performance of our algorithm can only depend on the particular sequence of sub- and super-median points in r. We can formalize this observation by defining the 0-1 sequence $s(r)$, where $s_i(r) = [r(i) > median(r)]$. Thus the performance of SUBMEDIAN-SEEKER will only depend on $s(r)$, and not on the specific values that r assumes. In other words, SUBMEDIAN-SEEKER only uses part of the information available to it in deciding where to search next.

It was hoped that $M(r)$ would completely describe the performance of SUBMEDIAN-SEEKER on a function r; however, simulation showed that even when $M(r)$ was fixed, there was a significant dependence on the structure of $s(r)$. This can be seen in Figure 1, where the distribution of SUBMEDIAN-SEEKER performance over a fixed $M(r)$ of 8 is shown[2].

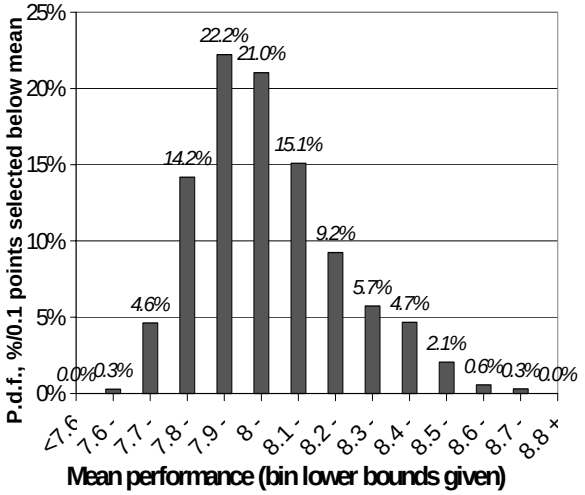

Figure 1: Distribution of mean performance of SUBMEDIAN-SEEKER on 5000 random functions with $n = 32$ and $M(r) = 8$. Performance is defined as the number of evaluated points (out of 16) whose values were below the mean. Mean performance were determined in each case by taking the mean of 1 000 000 runs of SUBMEDIAN-SEEKER. The standard error of the means ($2\ \sigma_{\bar{x}}$) are all less than 0.005.

[2] For details on how one generates random functions satisfying a given value of $M(r)$, see section 5, Appendix.

Consider a function with a small value for $M(r)$: what are its properties? Since $M(r)$ measures the number of submedian values of r that have successors with supermedian values, we know immediately that points below the median are likely to be followed by points below the median. Therefore, we might expect that an algorithm that uses sampling to determine where to go next to perform well on such functions. Indeed, this is observed in practice[3], as demonstrated in Figure 2, which is a representative scatter plot of mean performance obtained over 10 runs as we vary $M(r)$ from 0 to 15.

We might further expect the worst-case performance of SUBMEDIAN-SEEKER over many functions to be monotonic decreasing in $M(r)$; this too was observed (data not shown here). There must therefore be a "critical value" of M, below which all functions r in $\{r \in I_n \mid M(r) < M_{crit}\}$ will perform better than random search, on average.

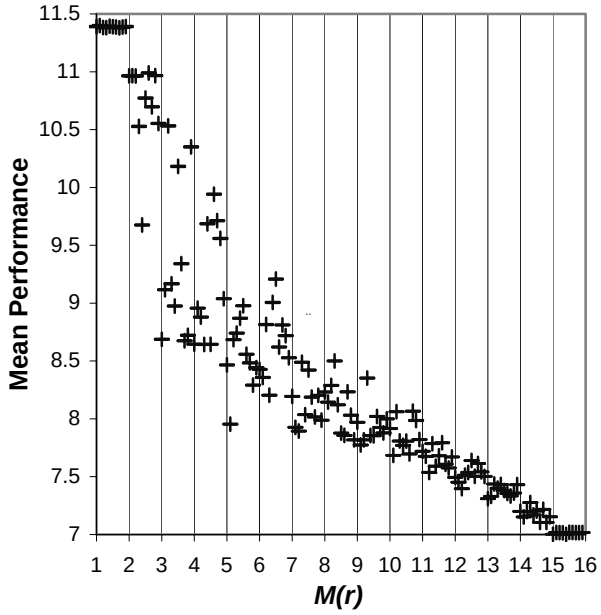

Figure 2: Average Performance of SUBMEDIAN-SEEKER with $n = 32$. Successive columns correspond to increasing values of $M(r)$; within a column, values are replicates with different randomly chosen functions. The standard errors of the means ($2\sigma_{\bar{x}}$) are all less than 0.009.

To understand the worst-case behaviour of SUBMEDIAN-SEEKER, we examined which patterns in s gave the poorest mean outcome. Careful experimentation, involving examining 10 000 random functions over 5 levels of M (not detailed here) revealed that the worst outcomes happened when the supermedian values were as unevenly distributed as possible, subject to the constraint that they occur in pairs; and when the submedian values were as symmetrically distributed as possible[4]. For instance, in the case where $n = 32$, $M = 3$, the worst outcomes occured with the pattern 12+; 6-; 2+; 5-; 2+; 5-.

Once the specific nature of this worst-case input was characterized, an algorithm was developed that generated and tested this worst case as the number of points in the function, n, and $M(r)$ were varied. In each case, 100 runs of SUBMEDIAN-SEEKER were performed, and a t-test of the hypothesis "the mean of this observed distribution is larger than that expected by chance" was performed at a significance level of α=0.001. If the t-test was inconclusive, the number of runs was doubled and the process repeated until it was conclusively proven either that the mean of the distribution is greater than or less than that expected by chance. A replicate of these data was obtained with α=0.000 01; no change in the results was observed. This test procedure was used to determine M_{crit}. This process was repeated for several values of n; the results are graphed in Figure 3.

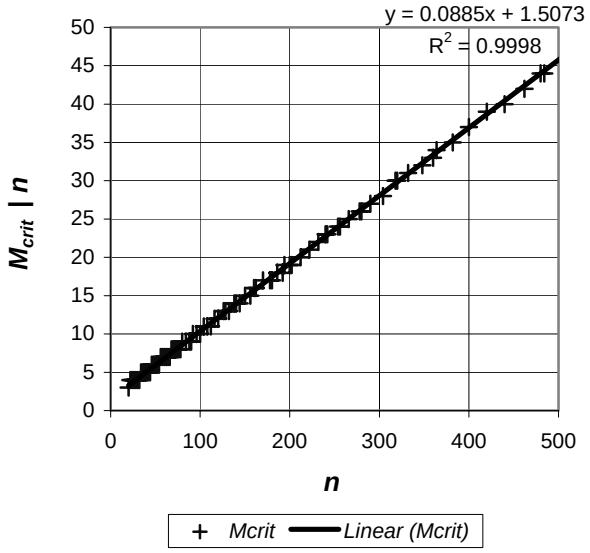

Figure 3: Graph of M_{crit} vs. n, for several values of n

As is evident from the graph, a clear linear trend to the data is observed. In fact, if we approximate the value of M_{crit} as $M_{crit} = \left\lfloor \frac{10n}{113} \right\rfloor + 1$[5], we are only incorrect in 2 of the cases experimentally sampled, wherein it is off by exactly 1.

[3] Note also that we would expect for random search to find that half of the sampled points fall below the median. For the case of figure 2, where 16 points were sampled, chance would lead us to expect 8 points below the median.

[4] The notation 12+; 6-; refers to the case where 12 consecutive points were above the median, followed by 6 below the median, and so on.

[5] See the ERRATA section, following the appendix (section 6)

We shall designate the quantity $\lim_{n\to\infty}\frac{M_{crit}}{n}$ as fr_{crit}. Note that the data do not permit selection of one particular value for fr_{crit}; 10/113ths is only the simplest such fraction that fits the data - all that is known is that the true multiplier is in the interval [0.088452, 0.088542].

Therefore, for all functions $r \in R \mid M(r) < M_{crit}$, we have an algorithm that can be expected to perform better than random chance at selecting points below the median. However, since the only property that we have used is that points in a function are above or below its median, the same argument applies for any function $f \in F$.

3 ANALYSIS

Such a result is only interesting if commonly searched functions have this property. We will show now that, because of the definition of $M(f)$ and SUBMEDIAN-SEEKER, many functions indeed do have this property.

If we restrict ourselves to 1-dimensional functions of a single real variable, we can say the following:

1. If $p \in P^k$, $p \neq 0$, is a uniformly sampled polynomial of degree at most k, and if $M_{crit} > k/2$, then SUBMEDIAN-SEEKER will perform better than random search on p.

Proof: There are most k solutions to $p(x) = b$. If we choose b = med p, then we know that there can be at most k median-crossings in total over the sampled interval, and specifically there can be at most $k/2$ crossings from submedian values of p to supermedian values of p. Since $M(p)$ measures exactly the number of such crossings, if $M(p) < M_{crit}$ then SUBMEDIAN-SEEKER will perform better than random search, on average, and the stated result follows.

2. If $f(x) = \sum_{j=0}^{k} a_j e^{-2\pi i j x}$ is a truncated Fourier series of at most k harmonics uniformly sampled over [0,1) at n locations, and if $M_{crit} > k/2$, then SUBMEDIAN-SEEKER will perform better than random search on f.

Proof: Write $f = \sum_{j=0}^{k} a_j e^{-2\pi i j x} = \sum_{j=0}^{k} a_j \left(e^{-2\pi i x}\right)^j$. This is a polynomial of degree k in $e^{-2\pi i x}$, and therefore can have at most k solutions for $e^{-2\pi i x} = b$. If we choose $e^{-2\pi i x}$ = med f, then we have that over the one period of the interval sampled, there can be at most k median crossings. The rest of the proof follows as in (1).

3. General valley case: If each extremum of a sampled function f is represented by $\frac{2}{fr_{crit}} \doteq \frac{113}{20}$ = 5.65… points on average[6], SUBMEDIAN-SEEKER will perform better than random search on f.

Proof: Suppose that there are k extrema of a sampled function. Since there can be at most 1 crossing of any central line between any pair of extrema, there can be at most $k/2$ crossings from submedian values of f to supermedian values of f. The rest of the proof follows as in (1).

4. Multidimensional functions: Let $p \in P_d^k(\bar{x})$ be a multivariate polynomial of dimension d and of degree at most k in each variable, regularly sampled l times in each dimension over a multidimensional rectangular interval. Assuming that we use the indexing function $In_p(\bar{x}) = \sum_{i=1}^{d} rank(\bar{x}_i) l^{i-1}$ to index the vector $\bar{x}$, if $k < \frac{2M_{crit}}{n} l - 2$, SUBMEDIAN-SEEKER will perform better than random search on p.

Proof: The values of the indexed function will vary most rapidly in the first dimension. There can therefore be at most $k/2$ crossings of the median from below for every sequence of l points from the polynomial, plus at most 1 more jump discontinuity in advancing the higher dimensional indices. So if $M_{crit} < l^{d-1}(\frac{k}{2}+1)$, SUBMEDIAN-SEEKER will outperform random search. Since $n = l^d$, the stated result follows.

The obvious extensions to multidimensional truncated Fourier series and limits on extrema also hold.

4 CONCLUSION

The No Free Lunch theorem has been very seriously considered and is very useful, especially in light of some of the sometimes-outrageous claims that had been made of specific optimization algorithms. However, once that theorem has been proven, the next logical step is to characterize how effective optimization can be under modest restrictions. We have operationally defined a technique for characterizing what makes a function "searchable". We have demonstrated the effectiveness of this technique by example - SUBMEDIAN-SEEKER will outperform random search over many functions of interest.

We emphasize that this is not merely one particular algorithm performing well in a particular domain; but instead a rather general and widely applicable set of conditions over which a given algorithm has been demonstrated to surpass random search. Let us demonstrate this point with an example. Consider the

[6] See the ERRATA section, following the appendix (section 6)

case of an optimization algorithm attempting to solve a multidimensional optimization problem using a vector of single-precision floating-point numbers as the domain. Such a representation has on the order of $l = 2^{23}$ points in any given octave in the domain. In particular, if we take the domain to be the hypercube $[x, 2x]^d$, then by (4) above, SUBMEDIAN-SEEKER will perform better than random search on any multivariate polynomial function of 1484707^{th} degree or lower.

We finally note that such modest non-uniformity of a source function is all that is needed for some optimization algorithms to outperform random search. While it may be useful and indeed is likely to improve efficiency to specialize a general-purpose algorithm towards the characteristics of a particular problem, this point is unrelated to the challenge posed by the No Free Lunch theorem.

Acknowledgments

The authors would like to thank the members of Carleton University's Artificial Life and Evolutionary Computing group for their valuable feedback during early versions of these results. Special thanks to Dr. S. Melkovian for his assistance in the proof of the Fourier result, and to Dr. Mark Wineberg for his advocacy of achieving understanding before publishing.

This work is supported in part by grants from the National Science and Engineering Research Council of Canada.

References

D.B. Fogel, *Evolutionary Computing: Toward a New Philosophy of Machine Intelligence*, Piscataway, New Jersey, IEEE Press, 1995.

R.L. Graham, D.E. Knuth, O. Patashnik, *Concrete Mathematics; a foundation for Computer Science*, 2nd Ed., Reading, Mass., Addison-Wesley, 1994.

W.G. Macready, D.H. Wolpert, *What Makes An Optimization Problem Hard,* SFI-TR-95-05-046, Oper. Res., (1996).

M. Mitchell, *An Introduction To Genetic Algorithms*, Cambridge, Mass., London, England, MIT Press, 1996.

J. Neter, M.H. Kutner, C.J. Nachtsheim, W. Wasserman, *Applied Linear Statistical Models*, 4th Ed., Boston, Mass., McGraw-Hill, 1996.

W.H. Press, *Numerical Recipes in C; the Art of Scientific Computing*, 2nd Ed., Cambridge, England, U.P., 1992.

H.-P. Schwefel, *Numerische Optimierung von Computermodellen mittels der Evolutionsstrategie*, Basel, Germany, Birkhaeuser, 1977.

D.H. Wolpert, W.G. Macready, *No Free Lunch Theorems For Search,* SFI-TR-95-02-010, Oper. Res., (1995).

5 APPENDIX

While the main text of the paper describes the experimental setup and experimentation thereon, there are a few details required to duplicate the experiments that the authors felt were best left outside of the main flow of the text. Inasmuch as is possible, we present those details here.

5.1 GENERATING AN UNBIASED RANDOM FUNCTION WITH A GIVEN *M(r)*

We consider only the case where $\delta = \rho = n$ (that is, the functions generated by this algorithm are invertible). We give a sketch of an algorithm to generate a random function (call it f) that has the property that $M(f)$ is the given input parameter.

By way of example:

Suppose that n is 32 and M is 4. There are therefore 4 points in r that are below the median that have successors that lie above the median. Consider the graph G of the values of f versus the values of their successors (as in Figure 4). We can characterize such a graph by noting that it divides nicely into 4 quadrants. Along the x-axis, there are points whose values are below the median, and points whose values are above the median. Along the y-axis, there are successor values both below and above the median.

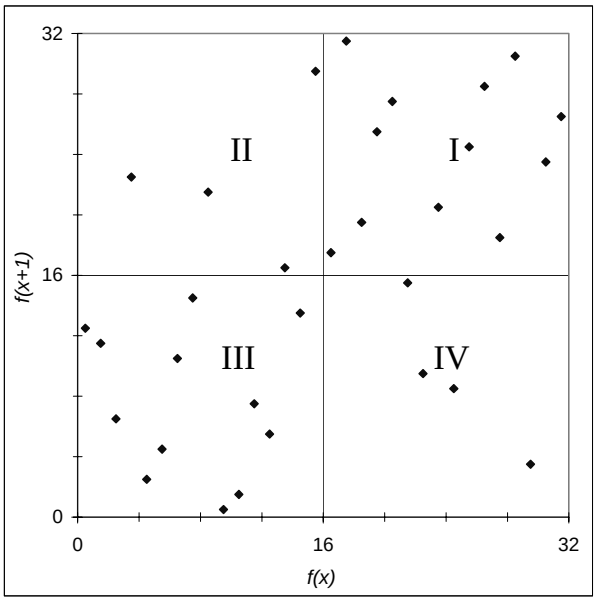

Figure 4: Sample graph of function values plotted against their successors, with $n = 32$ and $M = 4$.

Numbering the quadrants as usual in mathematics, we find that M describes the number of points in the second quadrant. By symmetry relations (exactly half the points are below the median), we can immediately see that there are $n - M$ points in quadrants I and III, and also M points in quadrant IV.

With these observations in hand, we can go about developing an algorithm[7] to randomly generate such a function that satisfies a particular value of M.

We go about generating such a function in reverse; first we develop the graph of $f(x)$ vs. $f(x+1)$, then we build a function from this graph.

We begin by placing the aforementioned number of points randomly in each quadrant. Coordinates are chosen at random; if any two points in any quadrant have the same x- or y-coordinate, we generate all the points anew. (It can be shown that this will occur with very low probability, if we have a modest value of n and we use 8-byte IEEE floating point numbers to store the coordinates).

We then replace each point's x- and y-coordinate by the relative ranks of their coordinates – thus converting the domain and range of our function to $\mathbb{N}_n$. We can represent this information as a vector which, for each value of $f(x)$, store $f(x+1)$. Call this vector v. At this point, it remains only to determine the values for the function f thus generated.

We choose the value for $f(0)$ at random from $[1, n]$.

We then look up successive values $f(1)$, $f(2)$... from our vector v. There are two possibilities:

- We encounter a cycle
- We encounter no cycle

If we don't encounter any cycles, we are done; so we continue assuming that there are cycles present.

We make a list of all the cycles that we've found, and of their consistuent members. For each cycle we find, we then attempt to break the cycle, while still preserving the property that exactly the desired number of points ends up in each quadrant in G.

For each member m of the cycle c, we try the following:

Note the quadrant of m. We then search for a point that lies in the same quadrant of G as m, but is not a member of the c. If we can find such a point (call it p), then swap m and p, and we've just linked two disjoint cycles. If no such point exists, try again with another member m of c. If we run out of members, then we have a cycle that cannot be merged with other cycles; at this point we give up and generate points randomly from the beginning.

As an optimization, after enumerating all the cycles, we attack the cycles in order of size, from smallest to largest. Since a smaller cycle has fewer elements that might be replaced, it is thought that this would improve overall performance, by forgoing analysis of those graphs that can not possibly generate a function.

6 ERRATA

(1) After submission of this paper, further experimentation revealed an error in the text of the paper. The critical fraction fr_{crit} is given in the paper as $fr_{crit} = \frac{10}{113}$. After additional consideration, it was realized that a lower bound on M_{crit} was required for the results given in the paper to hold. A systematic survey of $n \in \{20, 22, 24, ..., 902\}$ to find such a lower bound conclusively establishes the result $M_{crit} \geq \left\lfloor \frac{3n}{34} \right\rfloor + 1$.

Hence $fr_{crit} = \frac{3}{34} = 0.0882352...$

(2) Additionally, while preparing the presentation to accompany this paper, the author discovered that the equation for the general valley case was misprinted. The correct fraction is $\frac{1}{2\,fr_{crit}} = \frac{34}{6} = 5.66\overline{6}$.

[7] As an aside, it seems likely that elements of this algorithm have been published elsewhere. However, as the authors independently developed this technique to solve the present problem, we do not know to whom to attribute original discovery of such original elements. We would appreciate any guidance in this matter.