No Free Lunch Theorems for Search

David H. Wolpert(dhw@santafe.edu)
William G. Macready(wgm@santafe.edu)
The Santa Fe Institute
1399 Hyde Park Rd.
Santa Fe, NM, 87501, USA

February 6, 1995

Abstract

We show that all algorithms that search for an extremum of a cost function perform exactly the same, when averaged over all possible cost functions. In particular, if algorithm A outperforms algorithm B on some cost functions, then loosely speaking there must exist exactly as many other functions where B outperforms A. Starting from this we analyze a number of the other a priori characteristics of the search problem, like its geometry and its information-theoretic aspects. This analysis allows us to derive mathematical benchmarks for assessing a particular search algorithm's performance. We also investigate minimax aspects of the search problem, the validity of using characteristics of a partial search over a cost function to predict future behavior of the search algorithm on that cost function, and time-varying cost functions. We conclude with some discussion of the justifiability of biologically-inspired search methods.

1 Introduction

Many problems can be cast as optimization over a cost function. In such a problem, we are given a particular mapping $f : \mathcal{X} \rightarrow \mathcal{Y}$ ($\mathcal{F}$ being the set of all such mappings). For that f we seek the set of $x^* \in \mathcal{X}$ which give rise to a particular $y^* \in \mathcal{Y}$. Most often, we seek the x^*'s which extremize f (this will often be implicitly assumed in this paper). Physical examples of such a problem include free energy minimization ($\mathcal{Y} = \Re$) over spin configurations ($\mathcal{X} = \{-1, +1\}^N$), or over bond angles ($\mathcal{X} = \{\Re \times \Re \times \Re\}^N$), etc. Examples also abound in combinatorial optimization, ranging from number partitioning to graph coloring to scheduling [1].

There are two common approaches to these optimization problems. The first is a systematic construction of a good $\mathcal{X}$ value, x', from good sub-solutions

specifying part of x'. The most celebrated method of this type is the branch and bound algorithm [2]. For this systematic and exhaustive approach to work in reasonable time, one must have an effective heuristic, $h(n)$, representing the quality of sub-solutions n. There is extensive theoretical work [3] linking the cost function to the properties a heuristic must have in order to search efficiently.

A second approach to optimization begins with a population of one or more complete solutions $x \in \mathcal{X}$ and the associated $\mathcal{Y}$ values, and (tries to) iteratively improves upon those $\mathcal{X}$ values. There are many algorithms of this type, including hill-climbing, simulated annealing [4], and genetic algorithms [5].

Intuitively, one would expect that for this class of algorithms to work effectively, the biases in how they try to improve the population (i.e., the biases in how they search $\mathcal{X}$) must "match" those implicit in the cost function they are optimizing. However almost always these algorithms are directly applied, with little or no modification, to any cost function in a wide class of cost functions, with no concern for the particulars of the cost functions at hand. As we will demonstrate though, the "matching" intuition is true; blind faith in an algorithm to search effectively across a broad class of problems is rarely justified.

Indeed, one might expect that there are pairs of search algorithms A and B such that A performs better than B on average, even if B sometimes outperforms A. As an example, one might expect that hill-climbing usually outperforms hill-descending if one's goal is to find a maximum of the cost function. One might also expect it would outperform a random search. In point of fact though, as our central result demonstrates, this is not the case. If we do not take into account any particular biases or properties of our cost function, then the expected performance of all algorithms on that function are *exactly* the same.

In short, there are no "free lunches" for effective optimization; any algorithm performs only as well as the knowledge concerning the cost function put into the cost algorithm. For this reason we have dubbed our central result a "no free lunch" (NFL) theorem.

To prove the NFL theorem a framework has to be developed which addresses the core aspects of search. This framework constitutes the "skeleton" of the optimization problem; it is what can be said concerning search before explicit details of a particular real-world search problem are considered. The construction of such a skeleton provides a language to ask and answer formal questions about search, some of which have never before even been asked, never mind answered. (We pose and answer a number of such questions in this paper.) In addition, such a skeleton indicates where the real "meat" of optimization lies. It clarifies what the core issues are that underly the effectiveness of the search process.

The paper is organized as follows. We begin in Section 2 by presenting our framework and using it to prove the NFL theorem. We prove the theorem for both deterministic and stochastic search algorithms. Section 3 then gives a geometric interpretation of the NFL theorem. In particular, in that section we provide a geometric meaning of what it means for an algorithm to be well

"matched" to a cost function. It may be argued that the average behavior of algorithms is not an interesting quantity by which to compare algorithms, and thus the NFL results are of limited value. We address this potential criticism in Section 4 by investigating minimax distinctions between algorithms.Section 5 begins exploring some of the questions and answers raised by the framework developed in Section 2. Some of those answers lead naturally into results concerning the information theoretic aspects of search. Those results demonstrate the importance of the NFL theorem in analyzing optimization (those results are derived from the NFL theorem). A myriad of other properties of search may be investigated using techniques similar to those developed in this section. We list a sample of these in Section 9.2. In Section 6 we turn to the important problem of assessing the performance of particular search algorithms. We derive two "benchmarks" against which to compare such an algorithm's performance. Not all search problems are static; in many cases the cost function changes over time. Section 7 extends our analysis to the case of time dependent cost functions. In section 8 we provide some theorems valid for *any* single fixed cost function. These theorems state that one can not use a search algorithm's behavior so far for a particular cost function to predict its future behavior on that function. Finally, we conclude in Section 9 with the implications and future directions for our work.

The paper can be read in stages. A first reading might highlight the NFL theorem and its broad implications. Such a reading should start with Section 2 for an understanding of the NFL theorem, Eq. (1). Section 3 then provides a geometric understanding of the theorem. Section 4, which considers minimax distinctions between algorithms, addresses limitations of the NFL theorem. Finally, Section 9.1 discusses broad implications of the NFL result.

A second reading might explore the potential of the framework we have developed. Such a reading should include section 5, which uses our framework to demonstrate some of the information theoretic aspects of search. Section 6 then uses the framework to provide useful benchmarks against which other algorithms may be compared.

A final reading might investigate extensions of the above ideas. Such a reading would include section 7, which extends the NFL results to a class of time-dependent cost functions. It would also include section 8, which probes what may be learned from a limited amount of search over a single, specific, cost function. Concluding with Section 9.2 we list many directions for future extensions.

2 No Free Lunch Theorems for Search

All search algorithms rely on extrapolating from an existing set of m points and associated cost values, $(x, y)^m \in (\mathcal{X} \times \mathcal{Y})^m$, to a new, hopefully low cost point, $x' \in \mathcal{X}$. The extrapolation may be either deterministic or stochastic.

For simplicity take $\mathcal{X}$ and $\mathcal{Y}$ to be finite. Define $d_m \equiv \{d_m(i)\} \equiv \{d_m^x(i), d_m^y(i)\} \in \mathcal{D}_m$ for $i = 1 \ldots m$ to be a set of m distinct search points and associated cost values ordered in some way (usually according to the time at which they are generated) with the ordering index given by i. Let us call this a population of size m.

Let f indicate a single-valued function from $\mathcal{X}$ to $\mathcal{Y}$. Note that there are a finite number of f if $|\mathcal{X}|$ and $|\mathcal{Y}|$ are finite. At each stage of a search algorithm, a new point $x \in \mathcal{X}$ is chosen based on the preceding members of d; the pair $\{x', f(x')\}$ is added to d; and the procedure repeats.

Any search algorithm of the second type discussed in the introduction is a (perhaps probabilistic) mapping taking any population to a new point in the search space. For simplicity we assume that the new search point has not already been visited. (As discussed below, relaxing this assumption does not affect our results.) So in particular a deterministic search algorithm is a mapping $a : d \in \mathcal{D} \rightarrow \{x \mid x \notin d\}$, where $\mathcal{D} \equiv \cup_m \mathcal{D}_m$, and in particular contains the empty set. For clarity of the exposition, in this paper we will only explicitly consider such deterministic, non-retracing search algorithms, but, as discussed below, all our results also apply to stochastic and retracing algorithms.

We are interested in the histogram, $\vec{c}$, of cost values that an algorithm, a, obtains on a particular cost function, f, given m distinct cost evaluations. Note that $\vec{c}$ is given by the y values of the population, d_m^y, and is a vector of length $|\mathcal{Y}|$ whose ith component is the number of members in the population d_m having cost f_i. Once we have $\vec{c}$ we can use it to assess the quality of the search. For example if we are searching for minima we might take the minimum value in $\vec{c}$ as our performance measure. Consequently, we are interested in the conditional probability that histogram $\vec{c}$ will be obtained under m applications of algorithm a on f. We denote this quantity $P(\vec{c} \,|\, f, m, a)$.

A major result of this work is that $P(\vec{c} \,|\, f, m, a)$ is independent of a when we average over all cost functions. In other words,

Theorem: For any pair of algorithms a_1 and a_2,

$$\sum_f P(\vec{c} \,|\, f, m, a_1) = \sum_f P(\vec{c} \,|\, f, m, a_2). \tag{1}$$

An immediate consequence of this result is that the expected histograms, $E(\vec{c} \,|\, f, m, a) = \sum_{\vec{c}} \vec{c} P(\vec{c} \,|\, f, m, a)$, are on average identical between any two pairs of algorithms. More generally, at the point in their search where they have both created a population of size m, the performance of any two algorithms (measured for example as the depth of the minimum found) is, on average, identical (the average being over all possible cost functions). In particular if a_1 has better performance than a_2 over some subset $\phi \subset \mathcal{F}$ of functions, then a_2 must perform better on the set of remaining functions $\mathcal{F} \setminus \phi$. So for example if simulated annealing outperforms genetic algorithms on some set ϕ, genetic algorithms must outperform simulated annealing on $\mathcal{F} \setminus \phi$.

2.1 Proof for deterministic search

We now show that $\sum_f P(\vec{c}\,|\,f,m,a)$ has no dependence on a. Conceptually, the proof involves the following steps: First, we reduce the distribution over $\vec{c}$ values to one over d_m^y values. Then we use induction to establish the a-independence of the distribution over d_m^y. The inductive step starts by rearranging the distributions in question. Then f is broken up into two independent parts, one for $x \in d_m^x$ and one for $x \notin d_m^x$. These are evaluated separately, giving the desired result.

Expanding over all possible y components of a population of size m, d_m^y, we see

$$\sum_f P(\vec{c}\,|\,f,m,a) = \sum_{f,d_m^y} P(\vec{c},d_m^y\,|\,f,m,a)$$

Now $P(\vec{c},d_m^y\,|\,f,m) = P(\vec{c}\,|\,d_m^y,f,m,a)\,P(d_m^y\,|\,f,m,a)$. Moreover, the probability of obtaining a histogram $\vec{c}$ given f, d, m and a, $P(\vec{c}\,|\,d_m^y,f,m)$, depends only on the y values of population d_m. Therefore

$$\begin{aligned}\sum_f P(\vec{c}\,|\,f,m,a) &= \sum_{f,d_m^y} P(\vec{c}\,|\,d_m^y)\,P(d_m^y\,|\,f,m,a) \\ &= \sum_{d_m^y} P(\vec{c}\,|\,d_m^y) \sum_f P(d_m^y\,|\,f,m,a) \qquad (2)\end{aligned}$$

To prove that the expression in Eq. (2) is independent of a it suffices to show that for all m and d_m^y, $\sum_f P(d_m^y\,|\,f,m,a)$ is independent of a, since $P(\vec{c}\,|\,d_m^y)$ is independent of a. We will prove this by induction on m.

For $m = 1$ we write the population as $d_1 = \{d_1^x(a), f(d_1^x)\}$ where $d_1^x(a)$ is set by a. The only possible value for d_1^y is $f(x_1)$, so we have :

$$\sum_f P(d_1^y\,|\,f,m=1,a) = \sum_f \delta(d_1^y, f(d_1^x(a)))$$

where δ is the Kronecker delta function.

Now when we sum over all possible cost functions $\delta(d_1^y, f(d_1^x(a)))$ is 1 only for those functions which have cost d_1^y at point $d_1^x(a)$. Therefore that sum equals $|\mathcal{Y}|^{|\mathcal{X}|-1}$, independent of $d_1^x(a)$:

$$\sum_f P(d_1^y\,|\,f,m=1,a) = |\mathcal{Y}|^{|\mathcal{X}|-1}$$

which is independent of a. This bases the induction.

We now establish the inductive step, that if $\sum_f P(d_m^y\,|\,f,m,a)$ is independent of a for all d_m^y, then so also is $\sum_f P(d_{m+1}^y\,|\,f,m+1,a)$. This will complete the proof of the NFL result.

We start by writing

$$\begin{aligned} P(d_{m+1} \,|\, f, m+1, a) &= P(\{d_{m+1}(1), \ldots, d_{m+1}(m)\}, d_{m+1}(m+1) \,|\, f, m+1, a) \\ &= P(d_m, d_{m+1}(m+1) \,|\, f, m+1, a) \\ &= P(d_{m+1}(m+1) \,|\, d_m, f, m+1, a)\, P(d_m \,|\, f, m+1, a) \end{aligned}$$

so we have

$$\sum_f P(d^y_{m+1} \,|\, f, m+1, a) = \sum_f P(d^y_{m+1}(m+1) \,|\, d^y_m, f, m+1, a)\, P(d^y_m \,|\, f, m+1, a).$$

The new y value, $d^y_{m+1}(m+1)$, will depend on the new x value, f and nothing else. So we expand over these possible x values, getting

$$\begin{aligned} \sum_f P(d^y_{m+1} \,|\, f, m+1, a) &= \sum_{f,x} P(d^y_{m+1}(m+1) \,|\, f, x)\, P(x \,|\, d^y_m, f, m+1, a) \\ &\quad \times P(d^y_m \,|\, f, m+1, a) \\ &= \sum_{f,x} \delta(d^y_{m+1}(m+1), f(x))\, P(x \,|\, d^y_m, f, m+1, a) \\ &\quad \times P(d^y_m \,|\, f, m+1, a). \end{aligned}$$

Next note that since $x = a(d^x_m, d^y_m)$, it does not depend directly on f. Consequently we expand in d^x_m to remove the f dependence in $P(x \,|\, d^y_m, f, m+1, a)$:

$$\begin{aligned} \sum_f P(d^y_{m+1} \,|\, f, m+1, a) &= \sum_{f,x,d^x_m} \delta(d^y_{m+1}(m+1), f(x))\, P(x \,|\, d_m, a)\, P(d^x_m \,|\, d^y_m, f, m+1, a) \\ &\quad \times P(d^y_m \,|\, f, m+1, a) \\ &= \sum_{f,d^x_m} \delta(d^y_{m+1}(m+1), f(a(d_m)))\ \times P(d_m \,|\, f, m, a) \end{aligned}$$

where use was made of the fact that $P(x \,|\, d_m, a) = \delta(x, a(d_m))$ and the fact that $P(d_m \,|\, f, m+1, a) = P(d_m \,|\, f, m, a)$.

We do the sum over cost functions f first. The cost function is defined both over those points restricted to d^x_m and those points outside of d^x_m. $P(d_m \,|\, f, m)$ will depend on the f values defined over points inside d^x_m while $\delta(d^y_{m+1}(m+1), f(a(d_m)))$ depends only on the f values defined over points outside d^x_m. (Recall that $a(d^x_m) \notin d^x_m$.) So we have

$$\begin{aligned} \sum_f P(d^y_{m+1} \,|\, f, m+1, a) &= \sum_{d^x_m} \sum_{f(x \in d^x_m)} P(d_m \,|\, f, m, a) \\ &\quad \times \sum_{f(x \notin d^x_m)} \delta(d^y_{m+1}(m+1), f(a(d_m))). \end{aligned} \tag{3}$$

The sum $\sum_{f(x \notin d_m^x)}$ contributes a constant, $|\mathcal{Y}|^{|\mathcal{X}|-m-1}$, equal to the number of functions defined over points not in d_m^x passing through $(d_{m+1}^x(m+1), f(a(d_m)))$. So

$$\begin{aligned}
\sum_f P(d_{m+1}^y \,|\, f, m+1, a) &= |\mathcal{Y}|^{|\mathcal{X}|-m-1} \sum_{f(x \in d_m^x), d_m^x} P(d_m \,|\, f, m, a) \\
&= \frac{1}{|\mathcal{Y}|} \sum_{f, d_m^x} P(d_m \,|\, f, m, a) \\
&= \frac{1}{|\mathcal{Y}|} \sum_f P(d_m^y \,|\, f, m, a)
\end{aligned}$$

By hypothesis the right hand side of this equation is independent of a, so the left hand side must also be. This completes the proof of the NFL result.

Note that the no free lunch result implies that if we know nothing about f, then $P(\vec{c} \,|\, m, a)$, which is the probability we obtain histogram c after m distinct cost evaluations of algorithm a, is independent of a. This follows from

$$P(\vec{c} \,|\, m, a) = \sum_f P(\vec{c} \,|\, f, m, a)\, P(f \,|\, m, a) = \sum_f P(\vec{c} \,|\, f, m, a)\, P(f)$$

since the cost function doesn't depend on either m or a. If we know nothing about f then all f are equally likely, so for all f $P(f) = 1/|\mathcal{Y}|^{|\mathcal{X}|}$. (More generally, $P(f)$ reflects our "prior knowledge" concerning f.) Then $P(\vec{c} \,|\, m, a) = (1/|\mathcal{Y}|^{|\mathcal{X}|}) \sum_f P(c \,|\, f, m, a)$ which is independent of a by the no free lunch theorem.

The NFL theorem illustrates that even if we know something about f (perhaps specified through $P(f)$) but don't incorporate that knowledge into a then we have no assurances the a will be effective; we are simply relying on a fortuitous matching between f and a. This point is formally established in sections 3 and 8.

2.2 More general kinds of search

There are two restrictions on the definition of search algorithms used so far that one might find objectionable. These are: i) the banning of algorithms that might revisit the same points in $\mathcal{X}$ after placing them in d^x; and ii) the banning of algorithms that work stochastically rather than deterministically. Fortunately, the NFL result can easily be extended to include either algorithms that revisit points and/or are algorithms that are stochastic. So there is no loss of generality in our definition of a "search algorithm".

To see this, say we have a deterministic algorithm $a : d \in \mathcal{D} \to \{x \,|\, x \in \mathcal{X}\}$, so that given some (perhaps empty) d, the algorithm might produce a point $x \in d^x$. Call such an algorithm "potentially retracing". Given a potentially retracing

algorithm a, produce a new algorithm a' by "skipping over all duplications" in the sequence of $\{x, y\}$ pairs produced by the potentially retracing algorithm. Formally, for any d, $a'(d)$ is defined as the first x value from the sequence $\{a(\emptyset), a(d), a(a(d)), \ldots\}$ that is not contained in d^x. So long as the original algorithm a can not get stuck forever in some subset of d, we can always produce such an a' from a. (We can find no reason to design one's algorithm to not have an "escape mechanism" that ensures that it can not get stuck forever in some subset of d.) We will say that a' is a "compacted" version of a.

Now any two compacted algorithms are "search algorithms" in the sense the term is used in the previous subsection. Therefore they obey the NFL result of that subsection. So the NFL result in Eq. (1) holds even for potentially retracing algorithms, if we redefine 'm' in that equation to be the number of distinct points in the d^x's produced by the algorithms, in question, and if we redefine '$\vec{c}$' to be the histogram corresponding to those m distinct points.

Moreover, our real-world cost in using an algorithm is usually set by the number of distinct evaluations of $f(x)$. So it makes sense to compare potentially retracing algorithms not by looking at the d's they produce after being run the same number of times, but rather by looking at the d's they produce after sampling $f(x)$ the same number of times. This is consistent with using our redefined m and $\vec{c}$.

Note that the x at which a potentially retracing algorithm breaks out of a cycle might be stochastic (*e.g* simulated annealing). In this case the compacted version of the algorithm is still well-defined. Only rather than being deterministic, that compacted algorithm is stochastic. This brings us to the general issue of how to adapt our analysis to address stochastic search algorithms.

Let σ be a stochastic non-potentially retracting algorithm. Formally, this means that σ is a mapping taking any d to a (d-dependent) distribution over $\mathcal{X}$ that equals zero for all $x \in d^x$. So σ can be viewed as a "hyper-parameter", specifying the function $P(d^x_{m+1}(m+1) \,|\, d_m, \sigma)$ for all m and d.

Given this definition of σ, we can follow along with the derivation of the NFL result for deterministic algorithms, just with a replaced by σ throughout. Doing so, everything still holds. So that NFL result holds even for stochastic search algorithms. Therefore, by the same reasoning used to establish the no-free-lunch result for potentially retracing deterministic algorithms, the no-free-lunch result holds for potentially retracing stochastic algorithms.

3 A geometric interpretation

We can give a geometric interpretation of the no free lunch theorem by considering the space of possible cost functions. The probability of obtaining some histogram, $\vec{c}$, given m distinct cost evaluations using algorithm a is

$$P(\vec{c}\,|\,m,a) = \sum_f P(\vec{c}\,|\,m,a,f)P(f).$$

where $P(f)$ is the prior probability that the optimization problem at hand has cost function f. We can view the right-hand side of this equality as an inner product in $\mathcal{F}$:

Theorem: Define the $\mathcal{F}$-space vectors $\vec{v}_{c,a,m}$ and $\vec{p}$ by $\vec{v}_{c,a,m}(f) \equiv P(c\,|\,m,a,f)$ and $\vec{p}(f) \equiv P(f)$. Then

$$P(\vec{c}\,|\,m,a) = \vec{v}_{c,a,m} \cdot \vec{p} \tag{4}$$

This is an important equation. Any global knowledge you have about the properties of your cost function goes into the prior $\vec{p}$ over cost functions. $\vec{c}$ can be viewed as fixed to the histogram you want to obtain (usually one with a low cost value), and m is given by the constraints on the time we have to run our optimization algorithm. Thus the optimal algorithm is that which has the largest projection onto $\vec{p}$.

Taking this geometric view, the no free lunch result that $\sum_f P(\vec{c}\,|\,f,m,a)$ is independent of a has the simple interpretation that for a particular $\vec{c}$ and m, all algorithms a have the same projection onto the diagonal, that is $v_{c,a,m} \cdot \vec{1} = cst(\vec{c},m)$. For deterministic algorithms the components of $v_{c,a,m}$ (i.e., the probabilities that algorithm a gives histogram $\vec{c}$ on cost function f after m distinct cost evaluations) are all either 0 or 1 so the no free lunch result also implies $\sum_f P^2(\vec{c}\,|\,m,a,f) = cst(\vec{c},m)$. Geometrically, this means that the length of $\vec{v}_{c,a,m}$ is independent of a.

Thus all vectors $\vec{v}_{c,a,m}$ have the same length and lie on a cone with constant projection onto $\vec{1}$. Because the components of $\vec{v}_{c,a,m}$ are binary we might also view $\vec{v}_{c,a,m}$ as lying on the subset of the boolean hypercube having the same hamming distance from $\vec{0}$.

In Section 5we calculate two quantities concerning the distribution of $\vec{v}_{c,a,m}$ across vertices of this hypercube.

4 Minimax distinctions between algorithms

The NFL theorem does not address minimax properties of search. For example, say we're considering two deterministic algorithms, a_1 and a_2. It may very well be that there exist cost functions f such that a_1's histogram is much better (according to some appropriate quality measure) than a_2's, but no cost functions for which the reverse is true. For the NFL theorem to be obeyed in such a scenario, it would have to be true that there are many more f for which a_2's

algorithm is better than a_1's than vice-versa, but it is only slightly better for all those f. For such a scenario, in a certain sense a_1 has better minimax behavior than a_2; there are f for which a_1 beats a_2 badly, but none for which a_1 does substantially worse than a_2.

It appears though that analyzing minimax properties of algorithms is substantially more difficult than analyzing average behavior (like in the NFL theorem). Presently, nothing at all is known about minimax behavior involving stochastic algorithms. In particular, it is not known if in some sense a stochastic version of a deterministic algorithm has better/worse minimax behavior than that deterministic algorithm. In fact, even if we stick completely to deterministic algorithms, only an extremely preliminary understanding of minimax issues has been reached.

What we do know is the following. Consider the quantity

$$\sum_f P_{d^y_{m,1},d^y_{m,2}}(z, z' \mid f, m, a_1, a_2),$$

for deterministic algorithms a_1 and a_2. This quantity is just the number of f such that it is both true that a_1 produces a population with $\mathcal{Y}$ components z and that a_2 produces a population with $\mathcal{Y}$ components z'. In appendix B, it is proven that this quantity need not be symmetric under interchange of z and z':

Theorem: In general,

$$\sum_f P_{d^Y_{m,1},d^Y_{m,2}}(z, z' \mid f, m, a_1, a_2) \neq \sum_f P_{d^Y_{m,1},d^Y_{m,2}}(z', z \mid f, m, a_1, a_2). \qquad (5)$$

This means that under certain circumstances, even knowing only the $\mathcal{Y}$ components of the populations produced by two algorithms run on the same (unknown) f, we can infer something concerning what algorithm produced each population.

Now consider the quantity

$$\sum_f P_{C_1,C_2}(z, z' \mid f, m, a_1, a_2),$$

again for deterministic algorithms a_1 and a_2. This quantity is just the number of f such that it is both true that a_1 produces a histogram z and that a_2 produces a histogram z'. It too need not be symmetric under interchange of z and z' (see appendix B). This is a stronger statement then the asymmetry of d^Y's statement, since any particular histogram corresponds to multiple populations.

Amongst other things, currently nothing is known about "how big a problem" these asymmetries are. All of the asymmetries arise when the set of X values a_1 has visited overlaps with those that a_2 has visited. Given such overlap, and certain properties of how the algorithms generated the overlap, asymmetry arises. A precise specification of those "certain properties" is not yet in hand.

Nor is it known how generic they are, i.e., for what percentage of pairs of algorithms they arise. Although such issues are easy to state (see appendix B), it is not at all clear how best to answer them.

Note that neither of these two results directly address issues like whether there are f such that a_1's histogram is much better than a_2's, but not vice-versa. To answer that involves looking over all pairs of histograms such that there is the same relative quality between both histograms.

5 Information theoretic aspects of search

We first calculate the fraction of cost functions which give rise to a specific histogram $\vec{c}$ using algorithm a with m distinct cost points. This calculation allows us, for example, to answer the following question:

"What fraction of cost functions will give a particular distribution of cost values after m distinct cost evaluations chosen by using a genetic algorithm?"

This may seem an intractable question, but the NFL result allows us to answer it. It does this because it means that the fraction is independent of the algorithm! So we can answer the question by using an algorithm for which the calculation is particularly easy.

The algorithm we will use is one which visits points in $\mathcal{X}$ in some canonical order, say $x_1, x_2, \ldots, x_m$. Recall that the histogram $\vec{c}$ is specified by giving the frequencies of occurrence, across the $x_1, x_2, \ldots, x_m$, for each of the $|\mathcal{Y}|$ possible cost values.

Now the number of f's giving the desired histogram under our specified algorithm is just the multinomial giving the number of ways of distributing the cost values in $\vec{c}$. At the remaining $|\mathcal{X}| - m$ points in $\mathcal{X}$ the cost can assume any of the $|\mathcal{Y}|$ f values.

It will be convenient to define $\vec{\alpha} \equiv \frac{1}{m}\vec{c}$. Note that this is invariant if the contents of all bins in $\vec{c}$ are scaled by the same amount. By the argument of the preceding paragraph, the fraction we are interested in, $\rho_f(\vec{\alpha})$, is given by the following:

Theorem: For any algorithm, the fraction of cost functions that result in the histogram $\vec{c} = m\vec{\alpha}$ is given by

$$\rho_f(\vec{\alpha}) = \frac{\binom{m}{c_1\, c_2 \cdots c_{|\mathcal{Y}|}} |\mathcal{Y}|^{|\mathcal{X}|-m}}{|\mathcal{Y}|^{|\mathcal{X}|}} = \frac{\binom{m}{c_1\, c_2 \cdots c_{|\mathcal{Y}|}}}{|\mathcal{Y}|^m}. \tag{6}$$

Accordingly, $\rho_f(\vec{\alpha})$ can be related to the entropy of $\vec{c}$ in the standard way by using Stirling's approximation to order $\mathcal{O}(1/m)$, which is valid when all of the c_i are large:

$$\ln \binom{m}{c_1\, c_2\, \cdots\, c_{|\mathcal{Y}|}} = m \ln m - \sum_{i=1}^{|\mathcal{Y}|} c_i \ln c_i + \frac{1}{2}\Big[\ln m - \sum_{i=1}^{|\mathcal{Y}|} \ln c_i\Big]$$
$$= m\, S(\vec{\alpha}) + \frac{1}{2}\Big[\big(1 - |\mathcal{Y}|\big) \ln m - \sum_{i=1}^{|\mathcal{Y}|} \ln \alpha_i\Big]$$

where $S(\vec{\alpha}) = -\sum_{i=1}^{|\mathcal{Y}|} \alpha_i \ln \alpha_i$ is the entropy of the histogram $\vec{c}$. Thus for large enough m ($m \gg |\mathcal{Y}|$), the fraction of cost functions is given by the following formula:

Corollary:

$$\rho_f(\vec{\alpha}) \approx C(m, |\mathcal{Y}|)\, \frac{e^{m\, S(\vec{\alpha})}}{\prod_{i=1}^{|\mathcal{Y}|} \alpha_i^{1/2}}. \tag{7}$$

where $C(m, |\mathcal{Y}|)$ is a constant depending only on m and $|\mathcal{Y}|$.

If some of the $\vec{\alpha}_i$ are 0, Eq. (7) still holds, only with $\mathcal{Y}$ redefined to exclude the y's corresponding to the zero-valued $\vec{\alpha}_i$. However $\mathcal{Y}$ is defined, the normalization constant of Eq. (7) can be found by summing over all $\vec{\alpha}$ lying on the unit simplex. The details of such a calculation can be found in [8].

We next turn to a related question:

"On a given vertex of f-space (i.e., for a given cost function), what is the fraction of all algorithms that give rise to a particular $\vec{c}$?"

For this question, the only salient feature of f is its histogram (formed by looking across all $\mathcal{X}$) of cost values. Specify this histogram by $\vec{\beta}$; there are $N_i = \beta_i\, |\mathcal{X}|$ points in $\mathcal{X}$ for which $f(x)$ has the i'th $\mathcal{Y}$ value.

Call the fraction we are interested in $\rho_{\rm alg}(\vec{\alpha}, \vec{\beta})$. It turns out that $\rho_{\rm alg}(\vec{\alpha}, \vec{\beta})$ depends to leading order on the Kullback-Liebler "distance" [9] between $\vec{\alpha}$ and $\vec{\beta}$. To see this, we start with the following intuitively reasonable result, formally proven in appendix A:

Theorem: For a given f with histogram $\vec{N} = |\mathcal{X}|\vec{\beta}$, the fraction of algorithms that give rise to a histogram $\vec{c} = m\vec{\alpha}$ is given by

$$\rho_{\rm alg}(\vec{\alpha}, \vec{\beta}) = \frac{\prod_{i=1}^{|\mathcal{Y}|} \binom{N_i}{c_i}}{\binom{|\mathcal{X}|}{m}}. \tag{8}$$

The normalization factor in the denominator is simply the number of ways of selecting m cost values from $\mathcal{X}$.[1]

[1] It can also be determined from the identity $\sum_{\vec{c}} \delta(\sum_i c_i, m) \prod_i \binom{N_i}{c_i} = \binom{\sum_i N_i}{m}$.

The product of binomials can be approximated via Stirling's equation when both N_i and c_i are large:

$$\ln \prod_{i=1}^{|\mathcal{Y}|} \binom{N_i}{c_i} = \sum_{i=1}^{|\mathcal{Y}|} N_i \ln N_i - c_i \ln c_i - (N_i - c_i)\ln(N_i - c_i) + \ln N_i - \ln(N_i - c_i) - \ln c_i.$$

We assume $c_i/N_i \ll 1$, which is reasonable when $m \ll |\mathcal{X}|$. So using the expansion $\ln(1-z) = -z - z^2/2 - \ldots$, to second order in c_i/N_i we have

$$\ln \prod_{i=1}^{|\mathcal{Y}|} \binom{N_i}{c_i} = \sum_{i=1}^{|\mathcal{Y}|} c_i \ln(\frac{N_i}{c_i}) - \frac{1}{2}\ln c_i + c_i - \frac{c_i}{2N_i}\Big(c_i - 1 + (c_i - \frac{1}{2})(\frac{c_i}{N_i})^2 + \cdots\Big)$$

In terms of $\vec{\alpha}$ and $\vec{\beta}$ we finally obtain (using $m/|\mathcal{X}| \ll 1$)

$$\ln \prod_{i=1}^{|\mathcal{Y}|} \binom{N_i}{c_i} = mD_{KL}(\vec{\alpha}, \vec{\beta}) + m + |\mathcal{Y}| \ln(\frac{m}{|\mathcal{X}|}) - \sum_{i=1}^{|\mathcal{Y}|} \frac{1}{2}\ln(\alpha_i m) + \frac{m}{2|\mathcal{X}|}(\frac{\alpha_i}{\beta_i})(1 - \alpha_i m + \cdots),$$

where $D_{KL}(\vec{\alpha}, \vec{\beta}) \equiv -\sum_i \alpha_i \ln(\beta_i/\alpha_i)$ is the Kullbeck-Liebler distance between the distributions $\vec{\alpha}$ and $\vec{\beta}$.

Thus the fraction of algorithms is given by the following:

Corollary:

$$\rho_{\rm alg}(\vec{\alpha}, \vec{\beta}) \approx C(m, |\mathcal{X}|, |\mathcal{Y}|) \frac{{\rm e}^{-m\, D_{KL}(\vec{\alpha}, \vec{\beta})}}{\prod_{i=1}^{|\mathcal{Y}|} \alpha_i^{1/2}}. \tag{9}$$

where the constant C depends only on m, $|\mathcal{X}|$, and $|\mathcal{Y}|$.

As before, C can be calculated by summing $\vec{\alpha}$ over the unit simplex.

6 Measures of algorithm performance

In this section we calculate certain "benchmark" performance measures that allow us to assess the efficacy of any search algorithm.

Consider the case where low cost is preferable to high cost. Then in general we are interested in $P(\min(\vec{c}) > \epsilon \,|\, f, m, a)$, which is the probability that the

minimum cost an algorithm a finds in m distinct evaluations is larger than ϵ, given that the cost function is f. We consider three measures of an algorithm's performance that are related to this conditional probability:

i) The first measure is the average of this probability over all cost functions.

ii) The second is the form this conditional probability takes for the random algorithm, whose behavior is uncorrelated with the cost function.

iii) The third is the fraction of algorithms which, for a particular f and m, result in a $\vec{c}$ whose minimum exceeds ϵ.

These measures give us benchmarks which all truly "intelligent" algorithms should surpass when used in the real world; any algorithm that doesn't surpass them is doing a very poor job.

Recall that there are $|\mathcal{Y}|$ distinct cost values. With no loss of generality assume the i'th cost values equals i. So cost values run from a minimum of 0 to a maximum of $|\mathcal{Y}|$ in integer increments.

The first of our two benchmarks measures is

$$\frac{\sum_f P(\min(\vec{c}) > \epsilon \,|\, f, m, a)}{\sum_f 1} = \frac{\sum_{d^y_m, f} P(\min(d^y_m) > \epsilon \,|\, d^y_m)\, P(d^y_m \,|\, f, m, a)}{|\mathcal{Y}|^{|\mathcal{X}|}} \quad (10)$$

where in the last line we have marginalized over y values of populations of size m and noted that $\min(c) = \min(d^y_m)$.

Now consider $\sum_f P(d^y_m \mid f, m, a)$. The summand equals 0 or 1 for all f and deterministic a. In particular, it equals 1 if the following conditions are met

i) $f(d^x_m(1)) = d^y_m(1)$

ii) $f(a[d_m(1)]) = d^y_m(2)$

iii) $f(a[d_m(1), d_m(2)]) = d^y_m(3)$

...

These restrictions will always fix the value of $f(x)$ at exactly m points. f is completely free at all other points. Therefore

$$\sum_f P(d^y_m \mid f, m, a) = |\mathcal{Y}|^{|\mathcal{X}|-m}.$$

Using this result in Eq. (10) we find

$$\sum_f P(\min(\vec{c}) > \epsilon \,|\, f, m) \quad = \quad \frac{1}{|\mathcal{Y}|^m} \sum_{d^y_m} P((\min(d^y_m) > \epsilon \,|\, d^y_m)$$

$$= \frac{1}{|\mathcal{Y}|^m} \sum_{d_m^y \ni \min(d_m^y) > \epsilon} 1$$
$$= \frac{1}{|\mathcal{Y}|^m} (|\mathcal{Y}| - \epsilon)^m.$$

This establishes the following:

Theorem:

$$\sum_f P(\min(\vec{c}) > \epsilon \,|\, f, m) \quad = \quad \gamma^m(\epsilon). \tag{11}$$

where $\gamma(\epsilon) \equiv 1 - \epsilon/|\mathcal{Y}|$ is the fraction of cost lying above ϵ.

In a real world scenario, unless one's algorithm has its best-cost-so-far drop faster than this, there is no sense in which that algorithm is well-suited to searching the cost function at hand. The algorithm is doing no better than one would expect it to for a randomly chosen cost function.

Next we calculate the expected minimum of the cost values in the population as a function of m for the random algorithm, $\tilde{a}$, which picks points in $\mathcal{X}$ completely randomly, using no information from the current population. Marginalizing over histograms $\vec{c}$, the performance of $\tilde{a}$ is

$$P(\min(\vec{c}) \geq \epsilon \,|\, f, m, \tilde{a}) = \sum_{\vec{c}} P(\min(\vec{c}) \geq \epsilon \,|\, \vec{c})\, P(\vec{c} \,|\, f, m, \tilde{a})$$

Now $P(\vec{c} \,|\, f, m, \tilde{a})$ is the probability of obtaining histogram $\vec{c}$ in m random draws from the histogram $\vec{N}$ of the function f. (This can be viewed as the definition of $\tilde{a}$.) This probability has been calculated previously as $\frac{\prod_{i=1}^{|\mathcal{Y}|} \binom{N_i}{c_i}}{\binom{|\mathcal{X}|}{m}}$. So

$$\begin{aligned} P(\min(\vec{c}) \geq \epsilon \,|\, f, m, \tilde{a}) \quad &= \quad \frac{1}{\binom{|\mathcal{X}|}{m}} \sum_{c_1=0}^{m} \cdots \sum_{c_{|\mathcal{Y}|}=0}^{m} \delta(\sum_{i=1}^{|\mathcal{Y}|} c_i, m) P(\min(\vec{n}) > \epsilon \,|\, \vec{c}) \\ &\qquad \times \prod_{i=1}^{|\mathcal{Y}|} \binom{N_i}{c_i} \\ &= \quad \frac{1}{\binom{|\mathcal{X}|}{m}} \sum_{c_\epsilon=0}^{m} \cdots \sum_{c_{|\mathcal{Y}|}=0}^{m} \delta(\sum_{i=\epsilon}^{|\mathcal{Y}|} c_i, m) \prod_{i=\epsilon}^{|\mathcal{Y}|} \binom{N_i}{c_i} \\ &= \quad \frac{\binom{\sum_{i=\epsilon}^{|\mathcal{Y}|} N_i}{m}}{\binom{|\mathcal{X}|}{m}} \quad \text{(see footnote one)} \end{aligned}$$

$$= \quad \frac{\binom{\Gamma(\epsilon)|\mathcal{X}|}{m}}{\binom{|\mathcal{X}|}{m}} \tag{12}$$

This establishes the following:

Theorem: For the random algorithm $\tilde{a}$,

$$P(\min(\vec{c}) \geq \epsilon \,|\, f, m, \tilde{a}) \quad = \quad \prod_{i=0}^{m-1} \frac{\Gamma(\epsilon) - i/|\mathcal{X}|}{1 - i/|\mathcal{X}|}. \tag{13}$$

where $\Gamma(\epsilon) \equiv \sum_{i=\epsilon}^{|\mathcal{Y}|} N_i/|\mathcal{X}|$ is the fraction of points in $\mathcal{X}$ for which $f(x) \geq \epsilon$.

To first order in $1/|\mathcal{X}|$ this gives the following result:

Corollary:

$$P(\min(c) > \epsilon \,|\, f, m, \tilde{a}) = \Omega^m(\epsilon)\Big(1 - \frac{m(m-1)(1-\Omega(\epsilon))}{2\Omega(\epsilon)} \frac{1}{|\mathcal{X}|} + \ldots\Big). \tag{14}$$

This equation provides a useful benchmark against which any algorithm may be compared. Note in particular that for many cost functions cost values are distributed Gaussianly. For such a case, if the mean and variance of the Gaussian are μ and σ respectively, then $\Omega(\epsilon) = \mathrm{erfc}((\epsilon - \mu)/\sqrt{2}\sigma)/2$, where erfc is the complimentary error function.

Finally, to calculate the third performance measure, note that for fixed f and m, for any (deterministic) algorithm a, $P(\vec{c} > \epsilon \mid f, m, a)$ is either 1 or 0. Therefore the fraction of algorithms which result in a $\vec{c}$ whose minimum exceeds ϵ is given by

$$\frac{\sum_a P(\min(\vec{c}) > \epsilon \mid f, m, a)}{\sum_a 1}.$$

Expanding in terms of $\vec{c}$, we can rewrite the numerator of this ratio as $\sum_{\vec{c}} P(\min(\vec{c}) > \epsilon \mid \vec{c}) \quad \sum_a P(\vec{c} \mid f, m, a)$. However the ratio of this quantity to $\sum_a 1$ is exactly what we calculated when we evaluated measure ii) (see the beginning of the argument deriving Eq. (13)). This establishes the following:

Theorem: For fixed f and m, the fraction of algorithms which result in a $\vec{c}$ whose minimum exceeds ϵ is given by the quantity on the right-hand sides of Eqs. (13) and (14).

So in particular, consider the scenario where, when evaluated for ϵ equal to the minimum of the $\vec{c}$ produced in a particular run of your algorithm, the quantity given in Eq. (14) is less than 1/2. For such a scenario, your algorithm has done worse than over half of all search algorithms, for the f and m at hand.

7 Time-dependent cost functions

Here we establish a set of no free lunch results for a certain class of time-dependent cost functions. The time-dependent functions we are concerned with start with an initial cost function that is present when we sample the first x value. Then just before the beginning of each subsequent iteration of the search algorithm, the cost function is deformed to a new function, as specified by the mapping $T : \mathcal{F} \times \mathcal{N} \rightarrow \mathcal{F}$.[2] We write the function present during the sampling of the ith point as $f_{i+1} = T_i(f_i)$. We assume that at each step i, T_i is a bijection between $\mathcal{F}$ and $\mathcal{F}$. (Note the mapping induced by T from $\mathcal{F}$ to $\mathcal{F}$ can vary with the iteration number.) If this weren't the case, the evolution of cost functions could narrow in on a region of f's for which some algorithm, "by luck" as it were, happens to sample x values that lie near the extremizing x.

One difficulty with analyzing time-dependent cost functions is how to assess the quality of the search algorithm. In general there are two histogram-based schemes, involving two different populations of y values. As before, the population d_m^y is an ordered set of y values corresponding to the x values in d_m^x. The particular y value in d_m^y matching a particular x value in d_m^x is given by the cost function that was present when x was sampled. In contrast, the population D_m^y is defined to be the y values from the *present* cost function for each of the x values in d_m^x. Formally if $d_m^x = \{d_m^x(1), \cdots, d_m^x(m)\}$ then we have $d_m^y = \{f_1(d_m^x(1)), \cdots, T_{m-1}(f_{m-1})(d_m^x(m))\}$. Similarly, we have $D_m^y = \{T_{m-1}(f_{m-1})(d_m^x(1)), \cdots, T_{m-1}(f_{m-1})(d_m^x(m))\}$.

In some situations it may be that the members of the population "live" for a long time, on the time scale of the evolution of the cost function. In such situations it may be appropriate to judge the quality of the search algorithm with the histogram induced by D_m^y; all those previous elements of the population are still alive, and therefore their (current) fitness is of interest. On the other hand, if members of the population live for only a short time on the time scale of evolution of the cost function, one may instead be concerned with things like how well the living member(s) of the population track the changing cost function. In that kind of situation, it may make more sense to judge the quality of the search algorithm with the histogram induced by d_m^y.

Here we derive NFL results for both criteria. In analogy with the NFL theorem, we wish to average over all possible ways a cost function may be time-dependent, i.e., we wish to avenge over all T (rather than over all f, as in the NFL theorem). So consider the sum $\sum_T P(\vec{c}\ |, f_1, T, m, a)$ where f_1 is the initial cost function. Note first that since T only kicks in for $m > 1$, and since f_1 is fixed, there *are* a priori distinctions between algorithms as far as the first member of the population is concerned. So consider only histograms constructed from those elements of the population beyond the first. We will

[2] An obvious restriction would be to require that T doesn't vary with time, so that it is a mapping simply from $\mathcal{F}$ to $\mathcal{F}$. An analysis for T's limited this way is beyond the scope of this paper however.

prove the following:

Theorem: For all $\vec{c}$, $m > 1$, algorithms a_1 and a_2, and initial cost functions f_1,

$$\sum_T P(\vec{c} \,|\, f_1, T, m, a_1) = \sum_T P(\vec{c} \,|\, f_1, T, m, a_2). \tag{15}$$

We will show that this results holds whether $\vec{c}$ is constructed from d_m^y or from D_m^y. In analogy with the proof of the NFL theorem, we will do this by establishing the a-independence of $\sum_T P(\vec{c} \,|\, f, T, m, a)$.

We will begin by replacing each T in the sum with a set of cost functions, f_i, one for each iteration of the algorithm. To do this, we start with the following:

$$\begin{aligned}
\sum_T P(\vec{c} \,|\, f, T, m, a) &= \sum_T \sum_{d_m^x} \sum_{f_2 \cdots f_m} P(\vec{c} \mid \vec{f}, d_m^x, T, m, a) \\
&\quad \times P(f_2 \cdots f_m, d_m^x \mid f_1, T, m, a) \\
&= \sum_{d_m^x} \sum_{f_2 \cdots f_m} P(\vec{c} \mid \vec{f}, d_m^x) P(d_m^x \mid \vec{f}, m, a) \\
&\quad \times \sum_T P(f_2 \cdots f_m \mid f_1, T, m, a),
\end{aligned}$$

where we have indicated the sequence of cost functions, f_i, by the vector $\vec{f} = (f_1, \cdots, f_m)$.

Next we decompose the sum over all possible T into a series of sums. Each sum in the series is over the values T can take for one particular iteration of the algorithm. More formally, using $f_{i+1} = T_i(f_i)$, we write

$$\begin{aligned}
\sum_T P(\vec{c} \,|\, f, T, \, m \, , a) = &\sum_{d_m^x} \sum_{f_2 \cdots f_m} P(\vec{c} \mid \vec{f}, d_m^x) P(d_m^x \mid \vec{f}, m, a) \\
&\times \sum_{T_1} \delta(f_2, T_1(f_1)) \cdots \sum_{T_{m-1}} \delta(f_m, T_{m-1}(T_{m-2}(\cdots T_1(f_1)))).
\end{aligned}$$

(Note that $\sum_T P(\vec{c} \,|\, f, T, m, a)$ is independent of the values of $T_{i>m-1}$, so we can absorb those values into an overall a-independent proportionality constant.)

Now look at the innermost sum, over T_{m-1}, for some fixed values of the outer sum indices $T_1 \ldots T_{m-2}$. Now for fixed values of the outer sum indices $T_{m-1}(T_{m-2}(\cdots T_1(f_1)))$ is just some fixed cost function. Accordingly the innermost sum over T_{m-1} is simply the number of bijections of $\mathcal{F}$ that map that fixed cost function to f_m. This is just a constant, $(|\mathcal{F}| - 1)!$.

So we can do the T_{m-1} sum, and arrive at

$$\sum_T P(\vec{c} \,|\, f, T, \, m \, , a_1) \propto \sum_{d_m^x} \sum_{f_2 \cdots f_m} P(\vec{c} \mid \vec{f}, d_m^x) P(d_m^x \mid \vec{f}, m, a)$$

$$\times \sum_{T_1} \delta(f_2, T_1(f_1)) \cdots \sum_{T_{m-2}} \delta(f_{m-1}, T_{m-2}(T_{m-3}(\cdots T_1(f_1)))).$$

Now we can do the sum over T_{m-2}, in the exact same manner we just did the sum over T_{m-1}. In fact, all the sums over all T_i can be done, leaving us with

$$\begin{aligned} \sum_T P(\vec{c} \,|\, f, T, m\,, a_1) &\propto \sum_{d_m^x} \sum_{f_2 \cdots f_m} P(\vec{c} \mid \vec{f}, d_m^x) P(d_m^x \mid \vec{f}, m, a) \\ &= \sum_{d_m^x} \sum_{f_2 \cdots f_m} P(\vec{c} \mid \vec{f}, d_m^x) P(d_m^x \mid f_1 \cdots f_{m-1}, m, a). \quad (16) \end{aligned}$$

(In the last step we have exploited the statistical independence of d_m^x and f_m.)

To proceed further we must decide if we are interested in histograms formed from D_m^y or d_m^y. We begin with analysis of the D_m^y case. For this case $P(\vec{c} \mid \vec{f}, d_m^x) = P(\vec{c} \mid f_m, d_m^x)$, since D_m^y only reflects cost values from the last cost function, f_m. Plugging this in we get

$$\sum_T P(\vec{c} \,|\, f, T, m, a_1) \quad \propto \quad \sum_{d_m^x} \sum_{f_2 \cdots f_{m-1}} P(d_m^x \mid f_1 \cdots f_{m-1}, m, a) \sum_{f_m} P(\vec{c} \mid f_m, d_m^x)$$

The final sum over f_m is a constant equal to the number of ways of generating the histogram c from cost values drawn from f_m. This constant will involve the multinomial coefficient $\binom{m}{c_1 \cdots c_m}$ and some other factors. The important point is that it is independent of the particular d_m^x. Because of this we can evaluate the sum over d_m^x and thereby eliminate the a dependence.

$$\sum_T P(\vec{c} \,|\, f, T, m, a) \quad \propto \quad \sum_{f_2 \cdots f_{m-1}} \sum_{d_m^x} P(d_m^x \mid f_1 \cdots f_{m-1}, m, a) \propto 1$$

This completes the proof of Eq. (15) for the case where $\vec{c}$ is constructed from D_m^y.

Next we turn the case where we are interested not in D_m^y but in d_m^y. This case is considerably more difficult since we can not simplify $P(\vec{c} \mid \vec{f}, d_m^x)$ and thus can not decouple the sums over f_i. Nevertheless, the NFL result still holds. To see this we begin by expanding Eq. (16) over possible d_m^y values.

$$\begin{aligned} \sum_T P(\vec{c} \,|\, f, T, m, a) \quad \propto \quad & \sum_{d_m^x} \sum_{f_2 \cdots f_m} \sum_{d_m^y} P(\vec{c} \mid d_m^y) P(d_m^y \mid \vec{f}, d_m^x) \\ & \times P(d_m^x \mid f_1 \cdots f_{m-1}, m, a) \end{aligned}$$

$$= \sum_{d_m^y} P(\vec{c} \mid d_m^y) \sum_{d_m^x} \sum_{f_2 \cdots f_m} P(d_m^x \mid f_1 \cdots f_{m-1}, m, a) \times \prod_{i=1}^{m} \delta(d_m^y(i), f_i(d_m^x(i))) \quad (17)$$

The sum over the inner-most cost function, f_m, only has an effect on the $\delta(d_m^y(i), f_i(d_m^x(i)))$ term. So it contributes $\sum_{f_m} \delta(d_m^y(m), f_m(d_m^x(m)))$. This is a constant, equal to $|\mathcal{Y}|^{|\mathcal{X}|-1}$. We are left with

$$\sum_T P(\vec{c} \mid f, T, m, a) \propto \sum_{d_m^y} P(\vec{c} \mid d_m^y) \sum_{d_m^x} \sum_{f_2 \cdots f_{m-1}} P(d_m^x \mid f_1 \cdots f_{m-1}, m, a) \times \prod_{i=1}^{m-1} \delta(d_m^y(i), f_i(d_m^x(i))).$$

The sum over $d_m^x(m)$ is now trivial, so we have

$$\sum_T P(\vec{c} \mid f, T, m, a) \propto \sum_{d_m^y} P(\vec{c} \mid d_m^y) \sum_{d_m^x(1)} \cdots \sum_{d_m^x(m-1)} \sum_{f_2 \cdots f_{m-1}} P(d_{m-1}^x \mid f_1 \cdots f_{m-2}, m, a) \times \prod_{i=1}^{m-1} \delta(d_m^y(i), f_i(d_m^x(i))).$$

Now note that the above equation is of the exact same form as Eq. (17), only with a remaining population of size $m-1$ rather than m. Consequently, in an exactly analogous manner to the scheme we used to evaluate the sums over f_m and $d_m^x(m)$ that existed in Eq. (17), we can evaluate our sums over f_{m-1} and $d_m^x(m-1)$. Doing so simply generates more a-independent proportionality constants. Continuing in this manner, we evaluate all the sums over the f_i and arrive at

$$\sum_T P(\vec{c} \mid f, T, m, a_1) \propto \sum_{d_m^y} P(\vec{c} \mid d_m^y) \sum_{d_m^x(1)} P(d_m^x(1) \mid m, a) \, \delta(d_m^y(1), f_1(d_m^x(1))).$$

Now there is still algorithm-dependence in this result. However it is a trivial dependence; as previously discussed, it arises completely from how the algorithm selects the first x point in its population, $d_m^x(1)$. Since we consider only those points in the population that are generated *subsequent* to the first, our result says that there is no distinctions between algorithms. (Alternatively, we could consider all points in the population, even the first, and still get an NFL result, if in addition to summing over all T we sum over all f_1.) So even in the case

where we are interested in d_m^y the NFL result stills hold, subject to the minor caveats delineated above.

There are others way of assessing the quality of the search algorithm besides histograms based on D_m^y or d_m^y. For example, one may wish to not consider histograms at all; one may judge the quality of the search by the fitness of the most recent member of the population.

Similarly, there are other sums one could look at besides those over T. For example, one may wish to characterize what the aspects are of the relationship between a and T that determine $\sum_f P(\vec{c} \mid f, T, m, a)$. In fact, in general there *can* be a priori distinctions between algorithms as far as this quantity is concerned.

As an example of such distinctions, say that for all iterations of the search algorithm, T is the shift operator, replacing $f(x)$ by $f(x-1)$ for all x (with $\min(x) - 1 \equiv \max(x)$, and with $\mathcal{X}$ implicitly taken to be a contiguous set of integers). For this T, if a is the algorithm that first samples f at x_1, next at $x_1 + 1$, etc., regardless of the values in the population, then for any f, the histogram induced by d_m^y is always made up of identical $\mathcal{Y}$ values. Accordingly, $\sum_f P(\vec{c} \mid f, T, m, a) = 0$ for any $\vec{c}$ containing counts in more than one $\mathcal{Y}$ value bin. For other search algorithms, even for the same shift T, there is not this restriction on the set of allowed $\vec{c}$. So $\sum_f P(\vec{c} \mid f, T, m, a)$ is not independent of a in general.

Indeed, consider the shift same T, but used with a different algorithm, $\hat{a}$. This new algorithm looks at the $\mathcal{Y}$ value of the its first sample point x_1, and if that value is low, it samples at $x_1 + 1$, exactly like algorithm a. On the other hand, if that value is high, it samples some point other than $x_1 + 1$. In general, if one's goal is to find minimal $\mathcal{Y}$ values, $\hat{a}$ can be expected to outperform a, even when one averages over all f.

8 Fixed cost function results

One obvious difficulty with the NFL results discussed above is that one can always argue "oh, well in the real world $P(f)$ is not uniform, so the NFL results do not apply, and therefore I'm okay in using my favorite search algorithm". Of course, the premise does not follow from the proposition. Uniform $P(f)$ is a *typical* $P(f)$. (The uniform average of all $P(f)$ is the uniform $P(f)$.) So the actual $P(f)$ might just as easily be one for which your algorithm is poorly suited as one for which it is well suited. Ultimately, the only way to justify one's search algorithm is to argue in favor of a particular $P(f)$, and then argue that your algorithm is well suited to that $P(f)$. This is the only (!) legitimate way of defending a particular search algorithm against the implications of the NFL theorems.

Nonetheless, it is clearly of interest to derive NFL-type results that are inde-

pendent of $P(f)$. Certain such results apply to ways of choosing between search algorithms, and involve averaging over those search algorithms while keeping the cost function fixed. Although less sweeping than the NFL results, these results hold no matter what the real world's distribution over cost functions is.

Let a and a' be two search algorithms. Define a "choosing procedure" as one that examines two populations d and d', produced by a and a' respectively, and based on those populations, decides to use either a or a' for the subsequent part of the search. As an example, one choosing procedure is to choose a if and only the least cost element in d has lower cost than the least cost element in d'. As another example, a "stupid" choosing procedure would choose a if and only the least cost element in d has higher cost than the least cost element in d'.

At the point that you use a choosing procedure, you will have sampled the cost function at all the points in $d_\cup \equiv d \cup d'$. Accordingly, if $d_{>m}$ refers to the samples of the cost function that come after using the choosing algorithm, then the histogram the user is interested in is the histogram $c_{>m}$ which is the histogram formed from $d_{>m}$. In addition, for all the usual reasons, we can assume that the search algorithm chosen by the choosing procedure does not return to any points in $d_\cup$, without loss of generality[3].

The following theorem, proven in appendix C, tells us we have no *a priori* justification for using any particular choosing algorithm. Loosely speaking, no matter what the cost function, observing how well an algorithm has done so far tells us nothing about how well it would do if we continue to use it on the same cost function. (For simplicity, we only consider deterministic algorithms.)

Theorem: Let d and d' be two fixed populations both of size m, that are generated when the algorithms a and a' respectively are run on the cost function. Let A and B be two different choosing procedures. Let k be the number of elements in $c_{>m}$. Then

$$\sum_{a,a'} P(c_{>m} \,|\, f, d, d', k, a, a', A) = \sum_{a,a'} P(c_{>m} \,|\, f, d, d', k, a, a', B). \tag{18}$$

(It is implicit in this theorem that the sum excludes those algorithms a and a' that do not result in d and d' respectively when run on f.)

One might think that the preceding theorem is misleading, since it treats all populations equally, when for any given f some populations will be more likely

[3] a can know to avoid the elements *it* has seen before. However a priori, a has no way to avoid the elements it hasn't seen yet but that a' has (and vice-versa). Rather than have the definition of a somehow depend on the elements in $d' - d$ (and similarly for a'), we deal with this problem by defining $c_{>m}$ to be set only by those elements in $d_{>m}$ that lie outside of $d_\cup$. (This is similar to the procedure we developed above to deal with potentially retracing algorithms.) Formally, this means that the random variable $c_{>m}$ is a function of $d_\cup$ as well as of $d_{>m}$. It also means there may be fewer elements in the histogram $c_{>m}$ than there are in the population $d_{>m}$.

than others. However even if one weights populations according to their probability of occurrence, it is still true that, on average, the choosing procedure one uses has no effect on likely $c_{>m}$. This is established by the following corollary.

Corrolary: Under the conditions given in the preceding theorem,

$$\sum_{a,a'} P(c_{>m} \mid f, m, k, a, a', A) = \sum_{a,a'} P(c_{>m} \mid, f, m, k, a, a', B). \tag{19}$$

Proof: Let "*proc*" refer to our choosing procedure. We are interested in

$$\begin{aligned} \sum_{a,a'} P(c_{>m} \mid f, m, k, a, a', proc) \quad = \quad & \sum_{a,a',d,d'} P(c_{>m} \mid f, d, d', k, a, a', proc) \\ & \times P(d, d' \mid f, k, m, a, a', proc). \end{aligned}$$

Pull the sum over d and d' outside the sum over a and a'. Consider any term in that sum (i.e., any particular pair of values of d and d'). For that term, $P(d, d' \mid f, k, m, a, a', proc)$ is just 1 for those a and a' that result in d and d' respectively when run on f, and 0 otherwise. (Recall that we are assuming that a and a' are deterministic.) This means that the $P(d, d' \mid f, k, m, a, a', proc)$ factor simply restricts our sum over a and a' to the a and a' considered in our theorem. Accordingly, our theorem tell us that the summand of the sum over d and d' is the same for choosing procedures A and B. Therefore the full sum is the same for both procedures. QED.

These results tell us that there is no assumption for $P(f)$ that, by itself, justifies using some choosing procedure as far as subsequent search is concerned. To have an intelligent choosing procedure, one must take into account not only $P(f)$ but also the search algorithms one will be choosing among.

In fact, things may very well be worse than this. In supervised learning, there is a result related to the theorem above [10]. Translated into the current context that result suggests that if one restricts the sums to only be over those algorithms that are a good match to $P(f)$, then *stupid* choosing procedures – like choosing the algorithm with the less desirable $\vec{c}$ – outperform "smart" ones (which are the ones everyone uses in practice). An investigation of what exactly the set of algorithms summed over must be for a smart choosing procedure to be superior to a dumb one is beyond the scope of this paper. But clearly there are many subtle issues to disentangle.

9 Discussion and Future Work

9.1 Discussion

In this paper we present a framework for investigating search. This framework serves as a "skeleton" for the search problem; it tells us what we can know

about search before "fleshing in" the details of a particular real world search problem. Phrased differently, it provides a language in which to describe search algorithms, and in which to ask (and answer) questions about them.

Ultimately, of course, the only important question is, "How do I find good solutions for my given cost function f?" The proper answer to this question is to start with the given f, determine certain salient features of it, and then construct a search algorithm, a, specifically tailored to match those features. The inverse procedure - far more popular in some communities - is to investigate how specific algorithms perform on different f's. This inverse procedure is *only* of interest to the degree that it helps us with our primary procedure, of going from (features concerning) f to an appropriate a.

Note that often the "salient features" concerning f can be stated in terms of a distribution $P(f)$. To understand this, first note that we do in fact know f exactly. But at the same time, there is much about f that we need to know that is *effectively* unknown to us (e.g., f's extrema). In this, it is as though f is partially unknown. The very nature of the search process is to admit that you don't "know" f in full. As a result, it makes sense to (implicitly or otherwise) replace f with a distribution $P(f)$. In this, the search problem reduces to finding a good a for a particular $P(f)$ - exactly the issue addressed in Section 3 of this paper.

As an example of all this, it is well known that generic methods (like simulated annealing and genetic algorithms) are unable to compete with carefully hand-crafted solutions for specific search problems. The Traveling Salesman (TSP) problem is an excellent example of such a situation; the best search algorithms for the TSP problem are hand-tailored for it [12]. Linear programming problems are another example; the simplex algorithm is a search algorithm specifically designed to solve cost functions of a particular type. In both of these situations, the procedure followed by the researcher is to: identify salient aspects of f (e.g., it is a TSP problem, or it is a linear programming problem); *throw away all other knowledge concerning* f and thereby effectively replace f with a $P(f)$; and then use a search algorithm explicitly known to work well for that $P(f)$.

In our investigation of the search problem from this match-f-to-a perspective, the first question we addressed was whether it may be that some algorithm A performs better than B, on average. Our answer to this question, given by the NFL theorem is that this is impossible. An important implication of this result is the following. If a genetic algorithm outperforms simulated annealing (for example) over some class of cost functions Φ, then over the remaining cost functions $\mathcal{F} \setminus \Phi$, simulated annealing must outperform the genetic algorithm. It should be noted that this applies even if one considers "adaptive" search algorithms [6, 7] which modify their search strategy based on properties of the population of $(\mathcal{X} - - \mathcal{Y})$ pairs observed so far in the search, and which perform this "adaptation" without regard to any knowledge concerning salient features of f.

It is important to bear in mind exactly what all of this does (not) imply about the relationship between natural selection in the biological world and optimization (i.e. genetic algorithms). To this end, consider the extremely simplified view in which natural selection is viewed as optimization over a cost or "fitness" function. We further simplify matters by assuming the fitness function is static over time.

In this paper we measure an algorithm's performance based on *all* $\mathcal{X}$ values it has sampled since it began, and therefore we don't allow an algorithm to resample points it had already visited. Our NFL theorem states that all algorithms are equivalent by this measure. However one might consider different measures. In particular, we may be interested in the evolution through time of "generations" consisting of temporally contiguous subsets of our population, generations that are updated by our search algorithm. In such a scenario, it *does* make sense to resample points already visited. Moreover, our NFL theorem does not apply to this alternative kind of performance measure. For example, according to this alternative performance measure, an algorithm that resamples old points in $\mathcal{X}$ that are fit and adds them to the current generation will always do better than one that resamples old points that are not fit.

Now when we examine the biological world around us, we are implicitly using this second kind of measure; we only see the organisms from the current generation. In addition, natural selection means that only (essential characteristics of) good points in $\mathcal{X}$ are kept around from one generation to the next. Accordingly, using this second kind of performance measure, one expects that the average fitness across a generation improves with time. (Or would if the environment - i.e., cost function - didn't change in time, etc.) This is nothing more than the tautology that natural selection improves the fitness of the members of a generation.

However this empirical evidence that natural selection performs well according to this second measure does **not** mean anything concerning its performance according to the first measure. In particular, it does not mean that if we wish to do a search, and *are* able to keep around all points sampled so far, that we have any reason to believe that natural selection is an effective search strategy. Nor does it mean that natural selection works well as far as the tail of the measure based on the entire population is concerned. Yet it is precisely that tail that is of interest in the engineering world.

In short, the empirical evidence of the biological world does not indicate in any sense that natural selection is an effective search strategy, even in the biological world. We simply have not had a chance to observe the behavior of alternative strategies. For all we know, the strategy of breeding only the *least fit* members of the population may have done a better job at finding the extrema of the cost function faced by biological organisms. The experiment just has not been done. The breed-the-worst strategy will in general result in worse recent *generations*, but using that strategy implies nothing about the quality of the *populations* over the long term. If however, we relax the unrealistic assumption

that the fitness function is constant over time then it is possible that there may be disadvantages to a breed-the-worst policy.

To summarize, by the NFL theorem, any generation-based scheme that keeps only the worst members of the population for the next generation is equivalent to one that keeps the best members, on average. However, the fitness of the members of the generations will differ between the two search algorithms. This raises some obvious questions for future research: Averaged over all f, how big would one expect the difference to be? For a fixed f, and two random search algorithms that are "directed" differently in who they classify in being the current generation, how big would one expect the difference to be? How does this last calculation compare with the calculation made above of what the best member of the population will (likely) be for a random algorithm as m grows?

9.2 Future work

It is perhaps fitting for a paper about effective search that we conclude with a brief listing of other research directions we believe warrant further investigation.

The most important continuation of this work is to turn our framework into a practical tool to solve real problems. This would involve two steps. First we need a method of incorporating broad kinds of knowledge concerning f into the analysis. In this paper we have used $P(f)$ to do this, but perhaps there are other ways that we should also consider. For example, it is not yet clear how to (or even whether one should) encapsulate in a $P(f)$ the knowledge concerning the cost function that is implicit in the heuristics of Branch and Bound strategies. How do incorporate how the cost, f of a complete solution is accrued through the assemblage of sub-solutions?

The second step in this suggested program is to determine how best to convert knowledge concerning f into an optimal a. The goal in its broadest sense is to design a system that can take in such knowledge concerning f and then *solve* for the optimal a given that knowledge. One would then use that a to search the f.

In its fullest sense, this program may well involve many years of work. Nonetheless, there are many important questions related to this program that should be analyzable using only the tools developed in this paper. Many of them were presented in the text. Others, particularly well-suited to help us understand the connection between $P(f)$ and an optimal a, are: How fast does the cost histogram $\vec{c}$ associated with a particular algorithm converge to the histogram of the cost values f takes on across all of $\mathcal{X}$? As $P(f)$ changes from the diagonal in f space (i.e., from being uniform over all f), need some a's be hurt? Could the average over all a's improve? For what $P(f)$'s besides the diagonal are all algorithms equal? Given two particular algorithms (rather than all algorithms), for what $P(f)$ is the performance of the algorithms equal? In

particular, if $P(f)$ is uniform over some subset $\Phi \subset \mathcal{F}$ and zero outside Φ,[4] what are the equivalence classes of search algorithms with identical expected behavior?

Another interesting series of questions concerns differences between stochastic and deterministic algorithms. Are there potential advantages to stochastic algorithms? In particular, does it make sense to "expand" any stochastic algorithm σ in terms of deterministic algorithms a? I.e., can one write $P(c \mid f, m, \sigma) = \sum_a k_{a,\sigma} P(c|f, m, a)$ for some expansion coefficients $k_{a,\sigma}$? If so, it suggests that as $P(f)$ moves from the diagonal the performance of σ's will neither improve *nor degrade* as much as that of a's. So it may be that stochastic algorithms have certain minimax advantages over deterministic ones.

Acknowledgments
We would like to thank Unamay O'Reilly for helpful conversation, and the SFI for funding. Dhw would also like to thank TXN Inc. for funding.

References

[1] M.R. Garey, D.S. Johnson, *Computers and Intractability*, Freeman (1979).

[2] E.L. Lawler, D.E. Wood, *Operations Research*, **14**(4), 699-719, (1966).

[3] J. Pearl, *Heuristics, intelligent search strategies for computer problem solving*, Addison-Wesley, (1984).

[4] S. Kirkpatrick, C. D. Gelatt Jr., M. P. Vecchi, *Science*, **220**, 671, (1983).

[5] J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, (1975).

[6] L. Ingber, *Adaptive Simulated Annealing*, Software package documentation, `ftp.caltech.edu:/pub/ingber/asa.Z`.

[7] D. Yuret, M. de la Maza, Dynamic Hill-Climbing: Overcoming the limitations of optimization techniques in *The Second Turkish Symposium on Artificial Intelligence and Neural Networks*, pp208-212, (1993).

[8] C.E.M. Strauss, D.H. Wolpert, D.R. Wolf. Alpha, Evidence, and the Entropic Prior in *Maximum Entropy and Bayesian Methods*, ed. Ali Mohammed-Djafari, pp113-120, (1992).

[9] T. Cover, J. Thomas, *Elements of Information Theory*, John Wiley & Sons, (1991).

[4] As an example, Φ might be the set of correlated cost functions as in [13].

[10] D H. Wolpert, *Off-training set error and a priori distinctions between learning algorithms*, Technical Report SFI-TR-95-01-003, Santa Fe Institute, 1995.

[11] D H. Wolpert, *On Overfitting Avoidance as Bias*, Technical Report SFI-TR-92-03-5001, Santa Fe Institute, 1992.

[12] Gerhard Reinelt, *The Traveling Salesman, computational solutions for TSP applications*, Springer Verlag Berlin Heidelberg (1994).

[13] P.F. Stadler, *Europhys. Lett.* **20**, pp479-482, (1992).

A Proof related to information theoretic aspects of search

We want to calculate the proportion of all algorithms that give a particular $\vec{c}$ for a particular f. We proceed in several steps.

1) Since $\mathcal{X}$ is finite, populations are finite. Therefore any (deterministic) a is a huge - but finite - list. That list is indexed by all possible d's (aside from those that extend over the entire input space). Each entry in the list is the x the a in question outputs for that d-index.

2) Consider any particular unordered set of m $x-y$ pairs where no two of the pairs share the same x value. Such a set is an "unordered path" π. (Without loss of generality, from now on we implicitly restrict the discussion to unordered paths of length m.) A particular π is "in" or "from" a particular f if there is a unordered set of m $(x, f(x))$ pairs identical to π. The numerator on the right-hand side of Eq. (8) is the number of unordered paths in the given f that give the desired $\vec{c}$.

3) Claim: The number of unordered paths in f that give the desired $\vec{c}$ - the numerator on the right-hand side of Eq. (8) - is proportional to the number of a's that give the desired $\vec{c}$ for f. (The proof of this claim will constitute a proof of Eq. (8).) Furthermore, the proportionality constant is independent of f and $\vec{c}$.

4) Proof: We will construct a mapping $\phi : a \rightarrow \pi$. ϕ takes in an a that gives the desired $\vec{c}$ for f, and from it produces a π that is in f and gives the desired $\vec{c}$. We will then show that for any π the number of algorithms a such that $\phi(a) = \pi$ is a constant, independent of π, f, and $\vec{c}$. The proof will then be completed by showing that ϕ is single-valued, i.e., by showing that there is no a who has as image under mapping ϕ more than one π.

5) Any unordered path π gives a set of $m!$ different ordered paths in the obvious manner. (Note that every x value in an unordered path is distinct.) Each such ordered path π_{ord} in turn provides a set of m successive d's (if one includes the null d) and a following x. Indicate by $d(\pi_{ord})$ this set of the first

m d's provided by π_{ord}. (Note that any π_{ord} is itself a population, but to avoid confusion we avoid referring to it as such.)

6) For any ordered path π_{ord} we can construct a "partial algorithm". This consists of the list of an a, but with only the m $d(\pi_{ord})$ entries in the list filled in; the remaining entries are blank. (We say that m is the "length" of the partial algorithm.) Since there are $m!$ distinct partial a's for each π (one for each ordered path corresponding to π), we have $m!$ such partially filled-in lists for each π.

7) In the obvious manner we can talk about whether a particular partial algorithm is "consistent" with a particular full algorithm. This allows us to define (the inverse of) ϕ: for any π that is in f and gives $\vec{c}$, $\phi^{-1}(\pi) \equiv$ (the set of all a that are consistent with at least one partial algorithm generated from π and that give $\vec{c}$ when run on f).

8) To complete the first part of our proof we must show that for all π that are in f and give $\vec{c}$, $\phi^{-1}(\pi)$ contains the same number of elements, regardless of π, f, or c. To that end, first generate all ordered paths induced by π and then associate each such ordered path with a distinct m-element partial algorithm. Our question is how many full algorithms lists are consistent with at least one of these partial algorithm partial lists. (How this question is answered is the core of this appendix.)

9) To answer this question, reorder the entries in each of the partial algorithm lists by permuting the indices d of all the lists. Obviously such a reordering won't change the answer to our question.

9) We will perform the permuting by interchanging pairs of d indices. First, interchange any d index of the form $((d_X(1), d_Y(1)), \ldots, (d_X(i \leq m), d_Y(i \leq m)))$ whose entry is filled in in any of our partial algorithm lists with $d'(d) \equiv ((d_X(1), z), \ldots, (d_X(i), z))$, where z is some arbitrary constant $\mathcal{Y}$ value and x_j refers to the j'th element of $\mathcal{X}$. Next, create some arbitrary but fixed ordering of all $x \in \mathcal{X}$: $(x_1, \ldots, x_{|calX|})$. Then interchange any d' index of the form $((d_X(1), z, \ldots, (d_X(i \leq m), z)$ whose entry is filled in in any of our (new) partial algorithm lists with $d''(d') \equiv ((x_1, z), \ldots, (x_m, z))$. (Recall that all the $d_X(i)$ must be distinct.)

10) By construction, the resultant partial algorithm lists are independent of π, $\vec{c}$ and f, as is the number of such lists (it's $m!$). Therefore the number of algorithms consistent with at least one partial algorithm list in $\phi^{-1}(\pi)$ is independent of π, c and f. This completes the first part of the proof.

11) For the second part, first choose any 2 unordered paths that differ from one another, A and B. There is no ordered path A_{ord} constructed from A that equals an ordered path B_{ord} constructed from B. So choose any such A_{ord} and any such B_{ord}. If they disagree for the null d, then we know that there is no (deterministic) a that agrees with both of them. If they agree for the null d, then since they are sampled from the same f, they have the same single-element d. If they disagree for that d, then there is no a that agrees with both of them. If they agree for that d, then they have the same double-element d. Continue

in this manner all the up to the $(m-1)$-element d. Since the two ordered paths differ, they must have disagreed at some point by now, and therefore there is no a that agrees with both of them.

12) Since this is true for any A_{ord} from A and any B_{ord} from B, we see that there is no a in $\phi^{-1}(A)$ that is also in $\phi^{-1}(B)$. This completes the proof.

B Proof related to minimax distinctions between algorithms

The proofs are by example.

Consider three points in $\mathcal{X}$, x_1, x_2, and x_3, and three points in Y, y_1, y_2, and y_3.

1) Let the first point a_1 visits be x_1, and the first point a_2 visits be x_2.

2) If at its first point a_1 sees a y_1 or a y_2, it jumps to x_2. Otherwise it jumps to x_3.

3) If at its first point a_2 sees a y_1, it jumps to x_1. If it sees a y_2, it jumps to x_3.

Consider the cost function that has as the $\mathcal{Y}$ values for the three $\mathcal{X}$ values $\{y_1, y_2, y_3\}$, respectively.

For $m = 2$, a_1 will produce a population (y_1, y_2) for this function, and a_2 will produce (y_2, y_3).

The proof is completed if we show that there is no cost function so that a_1 produces a population containing y_2 and y_3 and such that a_2 produces a population containing y_1 and y_2.

There are four possible pairs of populations to consider:

i) $[(y_2, y_3), (y_1, y_2)]$;

ii) $[(y_2, y_3), (y_2, y_1)]$;

iii) $[(y_3, y_2), (y_1, y_2)]$;

iv) $[(y_3, y_2), (y_2, y_1)]$.

Since if its first point is a y_2 a_1 jumps to x_2 which is where a_2 starts, when a_1's first point is a y_2 its second point must equal a_2's second point. This rules out possibilities i) and ii).

For possibilities iii) and iv), by a_1's population we know that f must be of the form $\{y_3, s, y_2\}$, for some variable s. For case iii), s would need to equal y_1, due to the first point in a_2's population. However for that case, the second point a_2 sees would be the value at x_1, which is y_3, contrary to hypothesis.

For case iv), we know that the s would have to equal y_2, due to the first point in a_2's population. However that would mean that a_2 jumps to x_3 for its second point, and would therefore see a y_2, contrary to hypothesis.

Accordingly, none of the four cases is possible. This is a case both where there is no symmetry under exchange of d^y's between a_1 and a_2, and no symmetry under exchange of histograms. QED.

C Proof related to NFL results for fixed cost functions

Since any (deterministic) search algorithm is a mapping from $d \subset \mathcal{D}$ to $x \subset \mathcal{X}$, any search algorithm is a vector in the space $\mathcal{X}^{\mathcal{D}}$. The components of such a vector are indexed by the possible populations, and the value for each component is the x that the algorithm produces given the associated population.

Consider now a particular population d of size m. Given d, we can say whether any other population of size greater than m has the (ordered) elements of d as its first m (ordered) elements. The set of those populations that do start with d this way defines a set of components of any algorithm vector a. Those components will be indicated by $a_{\supseteq d}$.

The remaining components of a are of two types. The first is given by those populations that are equivalent to the first $M < m$ elements in d for some M. The values of those components for the vector algorithm a will be indicated by $a_{\subset d}$. The second type consists of those components corresponding to all remaining populations. Intuitively, these are populations that are not compatible with d. Some examples of such populations are populations that contain as one of their first m elements an element not found in d, and populations that re-order the elements found in d. The values of a for components of this second type will be indicated by $a_{\perp d}$.

Let *proc* be either A or B. We are interested in

$$\sum_{a, a'} P(c_{>m} \,|\, f, d_1, d_2, k\,, a, a', proc)$$

$$= \sum_{a_{\perp d}, a'_{\perp d'}} \sum_{a_{\subset d}, a'_{\subset d'}} \sum_{a_{\supseteq d}, a'_{\supseteq d'}} P(c_{>m} \,|\, f, d, d', k, a, a', proc).$$

The summand is independent of the values of $a_{\perp d}$ and $a'_{\perp d}$ for either of our two d's. In addition, the number of such values is a constant. (It is given by the product, over all populations not consistent with d, of the number of possible x each such population could be mapped to.) Therefore, up to an overall constant independent of d, d', f, and *proc*, our sum equals

$$\sum_{a_{\subset d}, a'_{\subset d'}} \sum_{a_{\supseteq d}, a'_{\supseteq d'}} P(c_{>m} \mid f, d, d', a_{\supseteq d}, a'_{\supseteq d'}, a_{\subset d}, a'_{\subset d'}, proc).$$

By definition, we are implicitly restricting the sum to those a and a' so that our summand is defined. This means that we actually only allow one value for each component in $a_{\subset d}$ (namely, the value that gives the next x element in d), and similarly for $a'_{\subset d'}$. Therefore our sum reduces to

$$\sum_{a_{\supseteq d}, a'_{\supseteq d'}} P(c_{>m} \mid f, d, d', a_{\supseteq d}, a'_{\supseteq d'}, proc).$$

Note that no component of $a_{\supseteq d}$ lies in $d^x_{\cup}$. The same is true of $a'_{\supseteq d'}$. So our sum over $a_{\supseteq d}$ is over the same components of a as the sum over $a'_{\supseteq d'}$ is of a'. Now for fixed d and d', *proc*'s choice of a or a' is fixed. Accordingly, without loss of generality, we can rewrite our sum as

$$\sum_{a_{\supseteq d}} P(c_{>m} \mid f, d, d', a_{\supseteq d}),$$

with the implicit assumption that $c_{>m}$ is set by $a_{\supseteq d}$. This sum is independent of *proc*. QED.