No Free Lunch Theorems for Optimization

David H. Wolpert
IBM Almaden Research Center
N5Na/D3
650 Harry Road
San Jose, CA 95120-6099

William G. Macready
Santa Fe Institute
1399 Hyde Park Road
Santa Fe, NM, 87501

December 31, 1996

Abstract

A framework is developed to explore the connection between effective optimization algorithms and the problems they are solving. A number of "no free lunch" (NFL) theorems are presented that establish that for any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class. These theorems result in a geometric interpretation of what it means for an algorithm to be well suited to an optimization problem. Applications of the NFL theorems to information theoretic aspects of optimization and benchmark measures of performance are also presented. Other issues addressed are time-varying optimization problems and *a priori* "head-to-head" minimax distinctions between optimization algorithms, distinctions that can obtain despite the NFL theorems' enforcing of a type of uniformity over all algorithms.

1 Introduction

The past few decades have seen increased interest in general-purpose "black-box" optimization algorithms that exploit little if any knowledge concerning the optimization problem on which they are run. In large part these algorithms have drawn inspiration from optimization processes that occur in nature. In particular, the two most popular black-box optimization strategies, evolutionary algorithms [FOW66, Hol93] and simulated annealing [KGV83] mimic processes in natural selection and statistical mechanics respectively.

In light of this interest in general-purpose optimization algorithms, it has become important to understand the relationship between how well an algorithm a performs and the optimization problem f on which it is run. In this paper we present a formal analysis that contributes towards such an understanding by addressing questions like the following: Given the plethora of black-box optimization algorithms and of optimization problems, how can we best match algorithms to problems (i.e., how best can we relax the black-box nature of the algorithms and have them exploit some knowledge concerning the optimization problem)? In particular, while serious optimization practitioners almost always perform such matching, it is usually on an ad hoc basis; how can such matching be formally analyzed? More generally, what is the underlying mathematical "skeleton" of optimization theory before the "flesh" of the probability distributions of a particular context and set of optimization problems are imposed? What can information theory and Bayesian analysis contribute to an understanding of these issues? How *a priori* generalizable are the performance results of a certain algorithm on a certain class of problems to its performance on other classes of problems? How should we even measure such generalization; how should we assess the performance of algorithms on problems so that we may programmatically compare those algorithms?

Broadly speaking, we take two approaches to these questions. First, we investigate what *a priori* restrictions there are on the pattern of performance of one or more algorithms as one runs over the set of all optimization problems. Our second approach is to instead focus on a particular problem and consider the effects of running over all algorithms. In the current paper we present results from both types of analyses but concentrate largely on the first approach. The reader is referred to the companion paper [MW96] for more kinds of analysis involving the second approach.

We begin in Section 2 by introducing the necessary notation. Also discussed in this section is the model of computation we adopt, its limitations, and the reasons we chose it.

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 such a context.

One of the main results of this paper is that such expectations are incorrect. We prove two NFL theorems in Section 3 that demonstrate this and more generally illuminate the connection between algorithms and problems. Roughly speaking, we show that for both static and time dependent optimization problems, the average performance of any pair of algorithms across all possible problems is exactly identical. This means in particular that if some algorithm a_1's performance is superior to that of another algorithm a_2 over some set of optimization problems, then the reverse must be true over the set of all other optimization problems. (The reader is urged to read this section carefully for a precise statement of these theorems.) This is true even if one of the algorithms is random; any algorithm a_1 performs *worse than randomly* just as readily (over the set of all optimization problems) as it performs better than randomly. Possible objections to these results are also addressed in Sections 3.1 and 3.2.

In Section 4 we present a geometric interpretation of the NFL theorems. In particular,

we show that an algorithm's average performance is determined by how "aligned" it is with the underlying probability distribution over optimization problems on which it is run. This Section is critical for anyone wishing to understand how the NFL results are consistent with the well-accepted fact that many search algorithms that do not take into account knowledge concerning the cost function work quite well in practice

Section 5.1 demonstrates that the NFL theorems allow one to answer a number of what would otherwise seem to be intractable questions. The implications of these answers for measures of algorithm performance and of how best to compare optimization algorithms are explored in Section 5.2.

In Section 6 we discuss some of the ways in which, despite the NFL theorems, algorithms can have *a priori* distinctions that hold even if nothing is specified concerning the optimization problems. In particular, we show that there can be "head-to-head" minimax distinctions between a pair of algorithms, it i.e., we show that considered one f at a time, a pair of algorithms may be distinguishable, even if they are not when one looks over all f's.

In Section 7 we present an introduction to the alternative approach to the formal analysis of optimization in which problems are held fixed and one looks at properties across the space of algorithms. Since these results hold in general, they hold for any and all optimization problems, and in this are independent of the what kinds of problems one is more or less likely to encounter in the real world. In particular, these results state that one has no *a priori* justification for using a search algorithm's behavior so far on a particular cost function to predict its future behavior on that function. In fact when choosing between algorithms based on their observed performance it does not suffice to make an assumption about the cost function; some (currently poorly understood) assumptions are also being made about how the algorithms in question are related to each other and to the cost function. In addition to presenting results not found in [MW96], this section serves as an introduction to perspective adopted in [MW96].

We conclude in Section 8 with a brief discussion, a summary of results, and a short list of open problems.

We have confined as many of our proofs to appendices as possible to facilitate the flow of the paper. A more detailed — and substantially longer — version of this paper, a version that also analyzes some issues not addressed in this paper, can be found in [WM95].

Finally, we cannot emphasize enough that **no claims whatsoever** are being made in this paper concerning how well various search algorithms work in practice. The focus of this paper is on what can be said *a priori*, without any assumptions and from mathematical principles alone, concerning the utility of a search algorithm.

2   Preliminaries

We restrict attention to combinatorial optimization in which the search space, $\mathcal{X}$, though perhaps quite large, is finite. We further assume that the space of possible "cost" values, $\mathcal{Y}$, is also finite. These restrictions are automatically met for optimization algorithms run on digital computers. For example, typically $\mathcal{Y}$ is some 32 or 64 bit representation of the real

numbers in such a case.

The size of the spaces $\mathcal{X}$ and $\mathcal{Y}$ are indicated by $|\mathcal{X}|$ and $|\mathcal{Y}|$ respectively. Optimization problems f (sometimes called "cost functions" or "objective functions" or "energy functions") are represented as mappings $f : \mathcal{X} \mapsto \mathcal{Y}$. $\mathcal{F} = \mathcal{Y}^{\mathcal{X}}$ is then the space of all possible problems. $\mathcal{F}$ is of size $|\mathcal{Y}|^{|\mathcal{X}|}$ — a very large but finite number. In addition to static f, we shall also be interested in optimization problems that depend explicitly on time. The extra notation needed for such time-dependent problems will be introduced as needed.

It is common in the optimization community to adopt an oracle-based view of computation. In this view, when assessing the performance of algorithms, results are stated in terms of the number of function evaluations required to find a certain solution. Unfortunately though, many optimization algorithms are wasteful of function evaluations. In particular, many algorithms do not remember where they have already searched and therefore often revisit the same points. Although any algorithm that is wasteful in this fashion can be made more efficient simply by remembering where it has been (c.f. tabu search [Glo89, Glo90]), many real-world algorithms elect not to employ this stratagem. Accordingly, from the point of view of the oracle-based performance measures, there are "artefacts" distorting the apparent relationship between many such real-world algorithms.

This difficulty is exacerbated by the fact that the amount of revisiting that occurs is a complicated function of both the algorithm and the optimization problem, and therefore cannot be simply "filtered out" of a mathematical analysis. Accordingly, we have elected to circumvent the problem entirely by comparing algorithms based on the number of *distinct* function evaluations they have performed. Note that this does not mean that we cannot compare algorithms that are wasteful of evaluations — it simply means that we compare algorithms by counting only their number of distinct calls to the oracle.

We call a time-ordered set of m distinct visited points a "sample" of size m. Samples are denoted by $d_m \equiv \{(d_m^x(1), d_m^y(1)), \cdots, (d_m^x(m), d_m^y(m))\}$. The points in a sample are ordered according to the time at which they were generated. Thus $d_m^x(i)$ indicates the $\mathcal{X}$ value of the ith successive element in a sample of size m and $d_m^y(i)$ is the associated cost or $\mathcal{Y}$ value. $d_m^y \equiv \{d_m^y(1), \cdots, d_m^y(m)\}$ will be used to indicate the ordered set of cost values. The space of all samples of size m is $\mathcal{D}_m = (\mathcal{X} \times \mathcal{Y})^m$ (so $d_m \in \mathcal{D}_m$) and the set of all possible samples of arbitrary size is $\mathcal{D} \equiv \cup_{m \geq 0} \mathcal{D}_m$.

As an important clarification of this definition, consider a hill-descending algorithm. This is the algorithm that examines a set of neighboring points in $\mathcal{X}$ and moves to the one having the lowest cost. The process is then iterated from the newly chosen point. (Often, implementations of hill-descending stop when they reach a local minimum, but they can easily be extended to run longer by randomly jumping to a new unvisited point once the neighborhood of a local minimum has been exhausted.) The point to note is that because a sample contains all the previous points at which the oracles was consulted, it includes the $(\mathcal{X}, \mathcal{Y})$ values of *all* the neighbors of the current point, and not only the lowest cost one that the algorithm moves to. This must be taken into account when counting the value of m.

Optimization algorithms a are represented as mappings from previously visited sets of points to a single new (*i.e.*, previously unvisited) point in $\mathcal{X}$. Formally, $a : d \in \mathcal{D} \mapsto \{x | x \notin d_{\mathcal{X}}\}$. Given our decision to only measure distinct function evaluations even if an

algorithm revisits previously searched points, our definition of an algorithm includes all common black-box optimization techniques like simulated annealing and evolutionary algorithms. (Techniques like branch and bound [LW66] are not included since they rely explicitly on the cost structure of partial solutions, and we are here interested primarily in black-box algorithms.)

As defined above, a search algorithm is deterministic; every sample maps to a unique new point. Of course essentially all algorithms implemented on computers are deterministic[1], and in this our definition is not restrictive. Nonetheless, it is worth noting that all of our results are extensible to non-deterministic algorithms, where the new point is chosen stochastically from the set of unvisited points. (This point is returned to below.)

Under the oracle-based model of computation any measure of the performance of an algorithm after m iterations is a function of the sample d_m^y. Such performance measures will be indicated by $\Phi(d_m^y)$. As an example, if we are trying to find a minimum of f, then a reasonable measure of the performance of a might be the value of the lowest $\mathcal{Y}$ value in d_m^y: $\Phi(d_m^y) = \min_i\{d_m^y(i) : i = 1\ldots m\}$. Note that measures of performance based on factors other than d_m^y (e.g., wall clock time) are outside the scope of our results.

We shall cast all of our results in terms of probability theory. We do so for three reasons. First, it allows simple generalization of our results to stochastic algorithms. Second, even when the setting is deterministic, probability theory provides a simple consistent framework in which to carry out proofs.

The third reason for using probability theory is perhaps the most interesting. A crucial factor in the probabilistic framework is the distribution $P(f) = P(f(x_1), \cdots, f(x_{|\mathcal{X}|}))$. This distribution, defined over $\mathcal{F}$, gives the probability that each $f \in \mathcal{F}$ is the actual optimization problem at hand. An approach based on this distribution has the immediate advantage that often knowledge of a problem is statistical in nature and this information may be easily encodable in $P(f)$. For example, Markov or Gibbs random field descriptions [KS80] of families of optimization problems express $P(f)$ exactly.

However exploiting $P(f)$ also has advantages even when we are presented with a single uniquely specified cost function. One such advantage is the fact that although it may be fully specified, many aspects of the cost function are *effectively* unknown (*e.g.*, we certainly do not know the extrema of the function.) It is in many ways most appropriate to have this effective ignorance reflected in the analysis as a probability distribution. More generally, we usually *act* as though the cost function is partially unknown. For example, we might use the same search algorithm for all cost functions in a class (*e.g.*, all traveling salesman problems having certain characteristics). In so doing, we are implicitly acknowledging that we consider distinctions between the cost functions in that class to be irrelevant or at least unexploitable. In this sense, even though we are presented with a single particular problem from that class, we act as though we are presented with a probability distribution over cost functions, a distribution that is non-zero only for members of that class of cost functions. $P(f)$ is thus a prior specification of the class of the optimization problem at hand, with different classes of problems corresponding to different choices of what algorithms we will

---

[1]In particular, note that random number generators are deterministic given a seed.

use, and giving rise to different distributions $P(f)$.

Given our choice to use probability theory, the performance of an algorithm a iterated m times on a cost function f is measured with $P(d_m^y|f, m, a)$. This is the conditional probability of obtaining a particular sample d_m under the stated conditions. From $P(d_m^y|f, m, a)$ performance measures $\Phi(d_m^y)$ can be found easily.

In the next section we will analyze $P(d_m^y|f, m, a)$, and in particular how it can vary with the algorithm a. Before proceeding with that analysis however, it is worth briefly noting that there are other formal approaches to the issues investigated in this paper. Perhaps the most prominent of these is the field of computational complexity. Unlike the approach taken in this paper, computational complexity mostly ignores the statistical nature of search, and concentrates instead on computational issues. Much (though by no means all) of computational complexity is concerned with physically unrealizable computational devices (Turing machines) and the worst case amount of resources they require to find optimal solutions. In contrast, the analysis in this paper does not concern itself with the computational engine used by the search algorithm, but rather concentrates exclusively on the underlying statistical nature of the search problem. In this the current probabilistic approach is complimentary to computational complexity. Future work involves combining our analysis of the statistical nature of search with practical concerns for computational resources.

3   The NFL theorems

In this section we analyze the connection between algorithms and cost functions. We have dubbed the associated results "No Free Lunch" (NFL) theorems because they demonstrate that if an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems. Additionally, the name emphasizes the parallel with similar results in supervised learning [Wol96a, Wol96b].

The precise question addressed in this section is: "How does the set of problems $F_1 \subset \mathcal{F}$ for which algorithm a_1 performs better than algorithm a_2 compare to the set $F_2 \subset \mathcal{F}$ for which the reverse is true?" To address this question we compare the sum over all f of $P(d_m^y|f, m, a_1)$ to the sum over all f of $P(d_m^y|f, m, a_2)$. This comparison constitutes a major result of this paper: $P(d_m^y|f, m, a)$ is independent of a when we average over all cost functions:

Theorem 1 *For any pair of algorithms a_1 and a_2,*

$$\sum_f P(d_m^y|f, m, a_1) = \sum_f P(d_m^y|f, m, a_2).$$

A proof of this result is found in Appendix A. An immediate corollary of this result is that for any performance measure $\Phi(d_m^y)$, the average over all f of $P(\Phi(d_m^y)|f, m, a)$ is independent of a. The precise way that the sample is mapped to a performance measure is unimportant.

This theorem explicitly demonstrates that what an algorithm gains in performance on one class of problems it necessarily pays for on the remaining problems; that is the only way that all algorithms can have the same f-averaged performance.

A result analogous to Theorem 1 holds for a class of time-dependent cost functions. The time-dependent functions we consider begin with an initial cost function f_1 that is present at the sampling of the first x value. Before the beginning of each subsequent iteration of the optimization algorithm, the cost function is deformed to a new function, as specified by a mapping $T : \mathcal{F} \times \mathcal{N} \to \mathcal{F}$.[2] We indicate this mapping with the notation T_i. So the function present during the ith iteration is $f_{i+1} = T_i(f_i)$. T_i is assumed to be a (potentially i-dependent) bijection between $\mathcal{F}$ and $\mathcal{F}$. We impose bijectivity because if it did not hold, the evolution of cost functions could narrow in on a region of f's for which some algorithms may perform better than others. This would constitute an *a priori* bias in favor of those algorithms, a bias whose analysis we wish to defer to future work.

How best to assess the quality of an algorithm's performance on time-dependent cost functions is not clear. Here we consider two schemes based on manipulations of the definition of the sample. In scheme 1 the particular $\mathcal{Y}$ value in $d_m^y(j)$ corresponding to a particular x value $d_m^x(j)$ is given by the cost function that was present when $d_m^x(j)$ was sampled. In contrast, for scheme 2 we imagine a sample D_m^y given by the $\mathcal{Y}$ 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 in scheme 1 we have $d_m^y = \{f_1(d_m^x(1)), \cdots, T_{m-1}(f_{m-1})(d_m^x(m))\}$, and in scheme 2 we have $D_m^y = \{f_m(d_m^x(1)), \cdots, f_m(d_m^x(m))\}$ where $f_m = T_{m-1}(f_{m-1})$ is the final cost function.

In some situations it may be that the members of the sample "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 by D_m^y; all those previous elements of the sample are still "alive" at time m, and therefore their current cost is of interest. On the other hand, if members of the sample 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 sample track the changing cost function. In such situations, it may make more sense to judge the quality of the algorithm with the d_m^y sample.

Results similar to Theorem 1 can be derived for both schemes. By analogy with that theorem, we average over all possible ways a cost function may be time-dependent, *i.e.*, we average over all T (rather than over all f). Thus we consider $\sum_T P(d_m^y | f_1, T, m, a)$ where f_1 is the initial cost function. Since T only takes effect 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. However after redefining samples to only contain those elements added after the first iteration of the algorithm, we arrive at the following result, proven in Appendix B:

Theorem 2 *For all d_m^y, D_m^y, $m > 1$, algorithms a_1 and a_2, and initial cost functions f_1,*

$$\sum_T P(d_m^y | f_1, T, m, a_1) = \sum_T P(d_m^y | f_1, T, m, a_2).$$

and

$$\sum_T P(D_m^y | f_1, T, m, a_1) = \sum_T P(D_m^y | f_1, T, m, a_2).$$

---

[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.

So in particular, if one algorithm outperforms another for certain kinds of evolution operators, then the reverse must be true on the set of all other evolution operators.

Although this particular result is similar to the NFL result for the static case, in general the time-dependent situation is more subtle. In particular, with time-dependence there are situations in which there can be *a priori* distinctions between algorithms even for those members of the population arising after the first. For example, in general there will be distinctions between algorithms when considering the quantity $\sum_f P(d_m^y | f, T, m, a)$. To see this, consider the case where $\mathcal{X}$ is a set of contiguous integers and for all iterations T is a shift operator, replacing $f(x)$ by $f(x-1)$ for all x (with $\min(x) - 1 \equiv \max(x)$). For such a case we can construct algorithms which behave differently *a priori*. For example, take a to be the algorithm that first samples f at x_1, next at $x_1 + 1$ and so on, regardless of the values in the population. Then for any f, d_m^y is always made up of identical $\mathcal{Y}$ values. Accordingly, $\sum_f P(d_m^y | f, T, m, a)$ is non-zero only for d_m^y for which all values $d_m^y(i)$ are identical. Other search algorithms, even for the same shift T, do not have this restriction on $\mathcal{Y}$ values. This constitutes an *a priori* distinction between algorithms.

3.1   Implications of the NFL theorems

As emphasized above, the NFL theorems mean that if an algorithm does particularly well on one class of problems then it most do more poorly over the remaining problems. In particular, if an algorithm performs better than random search on some class of problems then in must perform *worse than random search* on the remaining problems. Thus comparisons reporting the performance of a particular algorithm with particular parameter setting on a few sample problems are of limited utility. While sicj results do indicate behavior on the narrow range of problems considered, one should be very wary of trying to generalize those results to other problems.

Note though that the NFL theorem need not be viewed this way, as a way of comparing function classes $\mathcal{F}_1$ and $\mathcal{F}_2$ (or classes of evolution operators T_1 and T_2, as the case might be). It can be viewed instead as a statement concerning any algorithm's performance when f is not fixed, under the uniform prior over cost functions, $P(f) = 1/|\mathcal{F}|$. If we wish instead to analyze performance where f is not fixed, as in this alternative interprations of the NFL theorem, but in contrast with the NFL case f is now chosen from a non-uniform prior, then we must analyze explicitly the sum

$$P(d_m^y | m, a) = \sum_f P(d_m^y | f, m, a) P(f) \tag{1}$$

Since it is certainly true that any class of problems faced by a practitioner will not have a flat prior, what are the practical implications of the NFL theorems when viewed as a statement concerning an algorithm's performance for non-fixed f? This question is taken up in greater detail in Section 4 but we make a few comments here.

First, if the practitioner has knowledge of problem characteristics but does not incorporate them into the optimization algorithm, then $P(f)$ is effectively uniform. (Recall that

8

$P(f)$ can be viewed as a statement concerning the practitioner's choice of optimization algorithms.) In such a case, the NFL theorems establish that there are no formal assurances that the algorithm chosen will be at all effective.

Secondly, while most classes of problems will certainly have some structure which, if known, might be exploitable, the simple existence of that structure does not justify choice of a particular algorithm; that structure must be known and reflected directly in the choice of algorithm to serve as such a justification. In other words, the simple existence of structure *per se*, absent a specification of that structure, cannot provide a basis for preferring one algorithm over another. Formally, this is established by the existence of NFL-type theorems in which rather than average over specific cost functions f, one averages over specific "kinds of structure", *i.e.*, theorems in which one averages $P(d_m^y \mid m, a)$ over distributions $P(f)$. That such theorems hold when one averages over all $P(f)$ means that the indistinguishability of algorithms associated with uniform $P(f)$ is not some pathological, outlier case. Rather uniform $P(f)$ is a "typical" distribution as far as indistinguishability of algorithms is concerned. The simple fact that the $P(f)$ at hand is non-uniform cannot serve to determine one's choice of optimization algorithm.

Finally, it is important to emphasize that even if one is considering the case where f is not fixed, performing the associated average according to a uniform $P(f)$ is *not* essential for NFL to hold. NFL can also be demonstrated for a range of non-uniform priors. For example, any prior of the form $\prod_{x \in \mathcal{X}} P'(f(x))$ (where $P'(y = f(x))$ is the distribution of $\mathcal{Y}$ values) will also give NFL. The f-average can also enforce correlations between costs at different $\mathcal{X}$ values and NFL still obtain. For example if costs are rank ordered (with ties broken in some arbitrary way) and we sum only over all cost functions given by permutations of those orders, then NFL still holds.

The choice of uniform $P(f)$ was motivated more from theoretical rather pragramattic concerns, as a way of analyzing the theoretical structure of optimization. Nevertheless, the cautionary observations presented above make clear that an analysis of the uniform $P(f)$ case has a number of ramifications for practitioners.

3.2   Stochastic optimization algorithms

Thus far we have considered the case in which algorithms are deterministic. What is the situation for stochastic algorithms? As it turns out, NFL results hold even for such algorithms.

The proof of this is straightforward. Let σ be a stochastic "non-potentially revisiting" 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 $x \in d^x$. (In this sense σ is what in statistics community is known as a "hyper-parameter", specifying the function $P(d_{m+1}^x(m+1) \mid d_m, \sigma)$ for all m and d.) One can now reproduce the derivation of the NFL result for deterministic algorithms, only with a replaced by σ throughout. In so doing all steps in the proof remain valid. This establishes that NFL results apply to stochastic algorithms as well as deterministic ones.

4 A geometric perspective on the NFL theorems

Intuitively, the NFL theorem illustrates that even if knowledge of f (perhaps specified through $P(f)$) is not incorporated into a, then there are no formal assurances that a will be effective. Rather, effective optimization relies on a fortuitous matching between f and a. This point is formally established by viewing the NFL theorem from a geometric perspective.

Consider the space $\mathcal{F}$ of all possible cost functions. As previously discussed in regard to Equation (1) the probability of obtaining some d_m^y is

$$P(d_m^y|m, a) = \sum_f P(d_m^y|m, a, f)P(f).$$

where $P(f)$ is the prior probability that the optimization problem at hand has cost function f. This sum over functions can be viewed as an inner product in $\mathcal{F}$. More precisely, defining the $\mathcal{F}$-space vectors $\vec{v}_{d_m^y,a,m}$ and $\vec{p}$ by their f components $\vec{v}_{d_m^y,a,m}(f) \equiv P(d_m^y|m, a, f)$ and $\vec{p}(f) \equiv P(f)$ respectively,

$$P(d_m^y|m, a) = \vec{v}_{d_m^y,a,m} \cdot \vec{p}. \tag{2}$$

This equation provides a geometric interpretation of the optimization process. d_m^y can be viewed as fixed to the sample that is desired, usually one with a low cost value, and m is a measure of the computational resources that can be afforded. Any knowledge of the properties of the cost function goes into the prior over cost functions, $\vec{p}$. Then Equation (2) says the performance of an algorithm is determined by the magnitude of its projection onto $\vec{p}$, i.e. by how aligned $\vec{v}_{d_m^y,a,m}$ is with the problems $\vec{p}$. Alternatively, by averaging over d_m^y, it is easy to see that $E(d_m^y|m, a)$ is an inner product between $\vec{p}$ and $E(d_m^y|m, a, f)$. The expectation of any performance measure $\Phi(d_m^y)$ can be written similarly.

In any of these cases, $P(f)$ or $\vec{p}$ must "match" or be aligned with a to get desired behavior. This need for matching provides a new perspective on how certain algorithms can perform well in practice on specific kinds of problems. For example, it means that the years of research into the traveling salesman problem (TSP) have resulted in algorithms aligned with the (implicit) $\vec{p}$ describing traveling salesman problems of interest to TSP researchers.

Taking the geometric view, the NFL result that $\sum_f P(d_m^y|f, m, a)$ is independent of a has the interpretation that for any particular d_m^y and m, all algorithms a have the same projection onto the the uniform $P(f)$, represented by the diagonal vector $\vec{1}$. Formally, $v_{d_m^y,a,m} \cdot \vec{1} = cst(d_m^y, m)$. For deterministic algorithms the components of $v_{d_m^y,a,m}$ (i.e. the probabilities that algorithm a gives sample d_m^y on cost function f after m distinct cost evaluations) are all either 0 or 1, so NFL also implies that $\sum_f P^2(d_m^y \mid m, a, f) = cst(d_m^y, m)$. Geometrically, this indicates that the length of $\vec{v}_{d_m^y,a,m}$ is independent of a. Different algorithms thus generate different vectors $\vec{v}_{d_m^y,a,m}$ all having the same length and lying on a cone with constant projection onto $\vec{1}$. (A schematic of this situation is shown in Figure 1 for the case where $\mathcal{F}$ is 3-dimensional.) Because the components of $\vec{v}_{c,a,m}$ are binary we might equivalently view $\vec{v}_{d_m^y,a,m}$ as lying on the subset the vertices of the Boolean hypercube having the same hamming distance from $\vec{1}$.
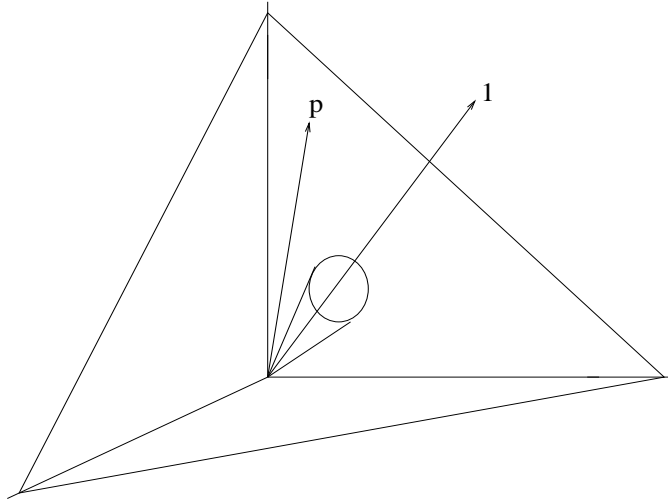
Figure 1: Schematic view of the situation in which function space $\mathcal{F}$ is 3-dimensional. The uniform prior over this space, $\vec{1}$ lies along the diagonal. Different algorithms a give different vectors v lying in the cone surrounding the diagonal. A particular problem is represented by its prior $\vec{p}$ lying on the simplex. The algorithm that will perform best will be the algorithm in the cone having the largest inner product with $\vec{p}$.

Now restrict attention to algorithms having the same probability of some particular d_m^y. The algorithms in this set lie in the intersection of 2 cones—one about the diagonal, set by the NFL theorem, and one set by having the same probability for d_m^y. This is in general an $|\mathcal{F}| - 2$ dimensional manifold. Continuing, as we impose yet more d_m^y-based restrictions on a set of algorithms, we will continue to reduce the dimensionality of the manifold by focusing on intersections of more and more cones.

The geometric view of optimization also suggests alternative measures for determining how "similar" two optimization algorithms are. Consider again Equation (2). In that the algorithm directly only gives $\vec{v}_{d_m^y,a,m}$, perhaps the most straight-forward way to compare two algorithms a_1 and a_2 would be by measuring how similar the vectors $\vec{v}_{d_m^y,a_1,m}$ and $\vec{v}_{d_m^y,a_2,m}$ are. (E.g., by evaluating the dot product of those vectors.) However those vectors occur on the right-hand side of Equation (2), whereas the performance of the algorithms — which is after all our ultimate concern — instead occur on the left-hand side. This suggests measuring the similarity of two algorithms not directly in terms of their vectors $\vec{v}_{d_m^y,a,m}$, but rather in terms of the dot products of those vectors with $\vec{p}$. For example, it may be the case that algorithms behave very similarly for certain $P(f)$ but are quite different for other $P(f)$. In many respects, knowing this about two algorithms is of more interest than knowing how their vectors $\vec{v}_{d_m^y,a,m}$ compare.

As another example of a similarity measure suggested by the geometric perspective, we could measure similarity between algorithms based on similarities between $P(f)$'s. For example, for two different algorithms, one can imagine solving for the $P(f)$ that optimizes

11

$P(d_m^y \mid m, a)$ for those algorithms, in some non-trivial sense.[3] We could then use some measure of distance between those two $P(f)$ distributions as a gauge of how similar the associated algorithms are.

Unfortunately, exploiting the inner product formula in practice, by going from a $P(f)$ to an algorithm optimal for that $P(f)$, appears to often be quite difficult. Indeed, even determining a plausible $P(f)$ for the situation at hand is often difficult. Consider, for example, TSP problems with N cities. To the degree that any practitioner attacks all N-city TSP cost functions with the same algorithm, that practitioner implicitly ignores distinctions between such cost functions. In this, that practitioner has implicitly agreed that the problem is one of how their fixed algorithm does across the set of all N-city TSP cost functions. However the detailed nature of the $P(f)$ that is uniform over this class of problems appears to be difficult to elucidate.

On the other hand, there is a growing body of work that does rely explicitly on enumeration of $P(f)$. For example, applications of Markov random fields [Gri76, KS80] to cost landscapes yield $P(f)$ directly as a Gibbs distribution.

5   Calculational applications of the NFL theorems

In this section we explore some of the applications of the NFL theorems for performing calculations concerning optimization. We will consider both calculations of practical and theoretical interest, and begin with calculations of theoretical interest, in which information-theoretic quantities arise naturally.

5.1   Information-theoretic aspects of optimization

For expository purposes, we simplify the discussion slightly by considering only the histogram of number of instances of each possible cost value produced by a run of an algorithm, and not the temporal order in which those cost values were generated. (Essentially all real-world performance measures are independent of such temporal information.) We indicate that histogram with the symbol $\vec{c}$; $\vec{c}$ has $\mathcal{Y}$ components $(c_{y_1}, c_{y_2}, \cdots, c_{y_{|y|}})$, where c_i is the number of times cost value $\mathcal{Y}_i$ occurs in the sample d_m^y.

Now consider any question like the following: "What fraction of cost functions give a particular histogram $\vec{c}$ of cost values after m distinct cost evaluations produced by using a particular instantiation of an evolutionary algorithm [FOW66, Hol93]?"

At first glance this seems to be an intractable question. However it turn out that the NFL theorem provides a way to answer it. This is because — according to the NFL theorem — the answer must be independent of the algorithm used to generate $\vec{c}$. Consequently we can chose an algorithm for which the calculation *is* tractable.

---

[3]In particular, one may want to impose restrictions on $P(f)$. For instance, one may wish to only consider $P(f)$ that are invariant under at least partial relabelling of the elements in $\mathcal{X}$, to preclude there being an algorithm that will assuredly "luck out" and land on $\min_{x \in \mathcal{X}} f(x)$ on its very first query.

Theorem 3 *For any algorithm, the fraction of cost functions that result in a particular histogram $\vec{c} = m\vec{\alpha}$ is*

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

For large enough m this can be approximated as

$$\rho_f(\vec{\alpha}) \cong C(m, |\mathcal{Y}|)\, \frac{\exp\left[m\, S(\vec{\alpha})\right]}{\prod_{i=1}^{|\mathcal{Y}|} \alpha_i^{1/2}}$$

where $S(\vec{\alpha})$ is the entropy of the distribution $\vec{\alpha}$, and $C(m, |\mathcal{Y}|)$ is a constant that does not depend on $\vec{\alpha}$.

This theorem is derived in Appendix C. If some of the $\vec{\alpha}_i$ are 0, the approximation 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 Equation (3) can be found by summing over all $\vec{\alpha}$ lying on the unit simplex [?].

A question related to one addressed in this theorem is the following: "For a given cost function, what is the fraction ρ_{alg} of all algorithms that give rise to a particular $\vec{c}$?" It turns out that the only feature of f relevant for this question is the histogram of its cost values formed by looking across all $\mathcal{X}$. Specify the fractional form of 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.

In Appendix D it is shown that to leading order, $\rho_{\mathrm{alg}}(\vec{\alpha}, \vec{\beta})$ depends on yet another information theoretic quantity, the Kullback-Liebler distance [CT91] between $\vec{\alpha}$ and $\vec{\beta}$:

Theorem 4 *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*

$$\rho_{\mathrm{alg}}(\vec{\alpha}, \vec{\beta}) = \frac{\prod_{i=1}^{|\mathcal{Y}|} \binom{N_i}{c_i}}{\binom{|\mathcal{X}|}{m}}. \tag{3}$$

For large enough m this can be written as

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

where $D_{KL}(\vec{\alpha}, \vec{\beta})$ is the Kullback-Liebler distnace between the distributions α and β.

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

5.2 Measures of performance

We now show how to apply the NFL framework to calculate certain benchmark performance measures. These allow both the programmatic (rather than ad hoc) assessment of the efficacy of any individual optimization algorithm and principled comparisons between algorithms.

Without loss of generality, assume that the goal of the search process is finding a minimum. So we are interested in the ϵ-dependence of $P(\min(\vec{c}) > \epsilon \mid f, m, a)$, by which we mean the probability that the minimum cost an algorithm a finds on problem f in m distinct evaluations is larger than ϵ. At least three quantities related to this conditional probability can be used to gauge an algorithm's performance in a particular optimization run:

i) The uniform average of $P(\min(\vec{c}) > \epsilon \mid f, m, a)$ over all cost functions.

ii) The form $P(\min(\vec{c}) > \epsilon \mid f, m, a)$ takes for the random algorithm, which uses no information from the sample d_m.

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

These measures give benchmarks which any algorithm run on a particular cost function should surpass if that algorithm is to be considered as having worked well for that cost function.

Without loss of generality assume the that i'th cost value (i.e., $\mathcal{Y}_i$ equals i. So cost values run from a minimum of 1 to a maximum of $|\mathcal{Y}|$, in integer increments. The following results are derived in Appendix E.

Theorem 5

$$\sum_f P(\min(\vec{c}) > \epsilon \mid f, m) = \omega^m(\epsilon)$$

where $\omega(\epsilon) \equiv 1 - \epsilon/|\mathcal{Y}|$ *is the fraction of cost lying above* ϵ. *In the limit of* $|\mathcal{Y}| \to \infty$, *this distribution obeys the following relationship*

$$\frac{\sum_f E(\min(\vec{c}) \mid f, m)}{|\mathcal{Y}|} = \frac{1}{m+1}.$$

Unless one's algorithm has its best-cost-so-far drop faster than the drop associated with these results, one would be hard-pressed indeed to claim that the algorithm is well-suited to the cost function at hand. After all, for such performance the algorithm is doing no better than one would expect it to for a randomly chosen cost function.

Unlike the preceding measure, the measures analyzed below take into account the actual cost function at hand. This is manifested in the dependance of the values of those measures on the vector $\vec{N}$ given by the cost function's histogram ($\vec{N} = |\mathcal{X}|\vec{\beta}$):

14

Theorem 6 *For the random algorithm $\tilde{a}$,*

$$P(\min(\vec{c}) \geq \epsilon \mid f, m, \tilde{a}) = \prod_{i=0}^{m-1} \frac{\Omega(\epsilon) - i/|\mathcal{X}|}{1 - i/|\mathcal{X}|}. \tag{4}$$

where $\Omega(\epsilon) \equiv \sum_{i=\epsilon}^{|\mathcal{Y}|} N_i/|\mathcal{X}|$ is the fraction of points in $\mathcal{X}$ for which $f(x) \geq \epsilon$. To first order in $1/|\mathcal{X}|$

$$P(\min(\vec{c}) > \epsilon \mid f, m, \tilde{a}) = \Omega^m(\epsilon)\left(1 - \frac{m(m-1)(1-\Omega(\epsilon))}{2\Omega(\epsilon)}\frac{1}{|\mathcal{X}|} + \cdots\right). \tag{5}$$

This result allows the calculation of other quantities of interest for measuring performance, for example the quantity

$$E(\min(\vec{c})|f, m, \tilde{a}) = \sum_{\epsilon=1}^{|\mathcal{Y}|} \epsilon \left[P(\min(\vec{c}) \geq \epsilon \mid f, m, \tilde{a}) - P(\min(\vec{c}) \geq \epsilon + 1 \mid f, m, \tilde{a})\right].$$

Note that for many cost functions of both practical and theoretical interest, cost values are distributed Gaussianly. For such cases, we can use that Gaussian nature of the distribution to facilitate our calculations. In particular, if the mean and variance of the Gaussian are μ and σ respectively, then we have $\Omega(\epsilon) = \text{erfc}((\epsilon - \mu)/\sqrt{2}\sigma)/2$, where erfc is the complimentary error function.

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 $\sum_{\vec{c}} P(\min(\vec{c}) > \epsilon|\vec{c}) \sum_a P(\vec{c} \mid f, m, a)$. However the ratio of this quantity to $\sum_a 1$ is exactly what was calculated when we evaluated measure (ii) (see the beginning of the argument deriving Equation (4)). This establishes the following:

Theorem 7 *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 Equations (4) and (5).*

As a particular example of applying this result, consider measuring the value of $\min(\vec{c})$ produced in a particular run of your algorithm. Then imagine that when it is evaluated for ϵ equal to this value, the quantity given in Equation (5) is less than $1/2$. In such a situation the algorithm in question has performaed worse than over half of all search algorithms, for the f and m at hand; hardly a stirring endorsement.

None of the discussion above explicitly concerns the dynamics of an algorithm's performance as m increases. Many aspects of such dynamics may be of interest. As an example, let

us consider whether, as m grows, there is any change in how well the algorithm's performance compares to that of the random algorithm.

To this end, let the sample generated by the algorithm a after m steps be d_m, and define $y' \equiv \min(d_m^y)$. Let k be the number of additional steps it takes the algorithm to find an x such that $f(x) < y'$. Now we can estimate the number of steps it would have taken the random search algorithm to search $\mathcal{X} - d_{\mathcal{X}}$ and find a point whose y was less than y'. The expected value of this number of steps is $1/z(d) - 1$, where $z(d)$ is the fraction of $\mathcal{X} - d_m^x$ for which $f(x) < y'$. Therefore $k + 1 - 1/z(d)$ is how much worse a did than would have the random algorithm, on average.

Next imagine letting a run for many steps over some fitness function f and plotting how well a did in comparison to the random algorithm on that run, as m increased. Consider the step where a finds its n'th new value of $\min(\vec{c})$. For that step, there is an associated k (the number of steps until the next $\min(d_m^y)$) and $z(d)$. Accordingly, indicate that step on our plot as the point $(n, k + 1 - 1/z(d))$. Put down as many points on our plot as there are successive values of $\min(\vec{c}(d))$ in the run of a over f.

If throughout the run a is always a better match to f than is the random search algorithm, then all the points in the plot will have their ordinate values lie below 0. If the random algorithm won for any of the comparisons though, that would mean a point lying above 0. In general, even if the points all lie to one side of 0, one would expect that as the search progresses there is corresponding (perhaps systematic) variation in how far away from 0 the points lie. That variation tells one when the algorithm is entering harder or easier parts of the search.

Note that even for a fixed f, by using different starting points for the algorithm one could generate many of these plots and then superimpose them. This allows a plot of the mean value of $k + 1 - 1/z(d)$ as a function of n along with an associated error bar. Similarly, one could replace the single number $z(d)$ characterizing the random algorithm with a full distribution over the number of required steps to find a new minimum. In these and similar ways, one can generate a more nuanced picture of an algorithm's performance than is provided by any of the single numbers given by the performance measure discussed above.

6   Minimax distinctions between algorithms

The NFL theorems do not direclty 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 performance 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 histogram 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 "head-to-head" 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.

Formally, we say that there exists head-to-head minimax distinctions between two algorithms a_1 and a_2 iff there exists a k such that for at least one cost function f, the difference $E(\vec{c} \mid f, m, a_1) - E(\vec{c} \mid f, m, a_2) = k$, but there is no other f for which $E(\vec{c} \mid f, m, a_2) - E(\vec{c} \mid f, m, a_1) = k$. (A similar definition can be used if one is instead interested in $\Phi(\vec{c})$ or d_m^y rather than $\vec{c}$.)

It appears that analyzing head-to-head minimax properties of algorithms is substantially more difficult than analyzing average behavior (like in the NFL theorem). Presently, very little is known about minimax behavior involving stochastic algorithms. In particular, it is not known if there are any senses in which 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

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

for deterministic algorithms a_1 and a_2. (By $P_A(a)$ is meant the distribution of a random variable A evaluated at $A = a$.) For deterministic algorithms, 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 F, it is proven by example that this quantity need not be symmetric under interchange of z and z':

Theorem 8 *In general,*

$$\sum_f P_{d_{m,1}^y, d_{m,2}^y}(z, z' \mid f, m, a_1, a_2) \neq \sum_f P_{d_{m,1}^y, d_{m,2}^y}(z', z \mid f, m, a_1, a_2). \qquad (6)$$

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 F). This is a stronger statement then the asymmetry of d^y's statement, since any particular histogram corresponds to multiple populations.

It would seem that neither of these two results directly implies that there are algorithms a_1 and a_2 such that for some f a_1's histogram is much better than a_2's, but for no f's is the reverse is true. To investigate this problem involves looking over all pairs of histograms (one

17

pair for each f) such that there is the same relationship between (the performances of the algorithms, as reflected in) the histograms. Simply having an inequality between the sums presented above does not seem to directly imply that the relative performances between the associated pair of histograms is asymmetric. (To formally establish this would involve creating scenarios in which there is an inequality between the sums, but no head-to-head minimax distinctions. Such an analysis is beyond the scope of this paper.)

On the other hand, having the sums equal does carry obvious implications for whether there are head-to-head minimax distinctions. For example, if both algorithms are deterministic, then for any particular f $P_{d^y_{m,1}, d^y_{m,2}}(z_1, z_2 \mid f, m, a_1, a_2)$ equals 1 for one (z_1, z_2) pair, and 0 for all others. In such a case, $\sum_f P_{d^y_{m,1}, d^y_{m,2}}(z_1, z_2 \mid f, m, a_1, a_2)$ is just the number of f that result in the pair (z_1, z_2). So $\sum_f P_{d^y_{m,1}, d^y_{m,2}}(z, z' \mid f, m, a_1, a_2) = \sum_f P_{d^y_{m,1}, d^y_{m,2}}(z', z \mid f, m, a_1, a_2)$ implies that there are no head-to-head minimax distinctions between a_1 and a_2. The converse does not appear to hold however.[4]

As a preliminary analysis of whether there can be head-to-head minimax distinctions, we can exploit the result in Appendix F, which concerns the case where $|\mathcal{X}| = |\mathcal{Y}| = 3$. First, define the following performance measures of two-element populations, $Q(d^y_2)$:

   i) $Q(y_2, y_3) = Q(y_3, y_2) = 2$.

   ii) $Q(y_1, y_2) = Q(y_2, y_1) = 0$.

   iii) Q of any other argument $= 1$.

In Appendix F we show that for this scenario there exist pairs of algorithms a_1 and a_2 such that for one f a_1 generates the histogram $\{y_1, y_2\}$ and a_2 generates the histogram $\{y_2, y_3\}$, but there is no f for which the reverse occurs (i.e., there is no f such that a_1 generates the histogram $\{y_2, y_3\}$ and a_2 generates $\{y_1, y_2\}$).

So in this scenario, with our defined performance measure, there *are* minimax distinctions between a_1 and a_2. For one f the performance measures of algorithms a_1 and a_2 are respectively 0 and 2. The difference in the Q values for the two algorithms is 2 for that f. However there are no other f for which the difference is -2. For this Q then, algorithm a_2 is minimax superior to algorithm a_1.

It is not currently known what restrictions on $Q(d^y_m)$ are needed for there to be minimax distinctions between the algorithms. As an example, it may well be that for $Q(d^y_m) = \min_i\{d^y_m(i)\}$ there are no minimax distinctions between algorithms.

More generally, at present nothing is known about "how big a problem" these kinds of asymmetries are. All of the examples of asymmetry considered here arise when the set of

---

[4]Consider the grid of all (z, z') pairs. Assign to each grid point the number of f that result in that grid point's (z, z') pair. Then our constraints are i) by the hypothesis that there are no head-to-head minimax distinctions, if grid point (z_1, z_2) is assigned a non-zero number, then so is (z_2, z_1); and ii) by the no-free-lunch theorem, the sum of all numbers in row z equals the sum of all numbers in column z. These two constraints do not appear to imply that the distribution of numbers is symmetric under interchange of rows and columns. Although again, like before, to formally establish this point would involve explicitly creating search scenarios in which it holds.

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 F), it is not at all clear how best to answer them.

However consider the case where we are assured that in m steps the populations of two particular algorithms have not overlapped. Such assurances hold, for example, if we are comparing two hill-climbing algorithms that start far apart (on the scale of m) in $\mathcal{X}$. It turns out that given such assurances, there are no asymmetries between the two algorithms for m-element populations. To see this formally, go through the argument used to prove the NFL theorem, but apply that argument to the quantity $\sum_f P_{d^y_{m,1}, d^y_{m,2}}(z, z' \mid f, m, a_1, a_2)$ rather than $P(\vec{c} \mid f, m, a)$. Doing this establishes the following:

Theorem: If there is no overlap between $d^x_{m,1}$ and $d^x_{m,2}$, then

$$\sum_f P_{d^y_{m,1}, d^y_{m,2}}(z, z' \mid f, m, a_1, a_2) = \sum_f P_{d^y_{m,1}, d^y_{m,2}}(z', z \mid f, m, a_1, a_2). \tag{7}$$

An immediate consequence of this theorem is that under the no-overlap conditions, the quantity $\sum_f P_{C_1, C_2}(z, z' \mid f, m, a_1, a_2)$ is symmetric under interchange of z and z', as are all distributions determined from this one over C_1 and C_2 (e.g., the distribution over the difference between those C's extrema).

Note that with stochastic algorithms, if they give non-zero probability to all d^x_m, there is always overlap to consider. So there is always the possibility of asymmetry between algorithms if one of them is stochastic.

7 $P(f)$-independent results

All work to this point has largely considered the behavior of various algorithms across a wide range of problems. In this section we introduce the kinds of results that can be obtained when we reverse roles and consider the properties of many algorithms on a *single* problem. More results of this type are found in [MW96]. The results of this section, although less sweeping than the NFL 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 a rule that examines the samples d_m and d'_m, 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 "rational" choosing procedure is to use a for the subsequent part of the search if and only it has generated a lower cost value in its sample than has a'. Conversely we can consider a "irrational" choosing procedure that went with the algorithm that had *not* generated the sample with the lowest cost solution.

At the point that a choosing procedure takes effect the cost function will have been sampled at $d_\cup \equiv d_m \cup d'_m$. Accordingly, if $d_{>m}$ refers to the samples of the cost function that

come after using the choosing algorithm, then the user is interested in the remaining sample $d_{>m}$. As always, without loss of generality it is assumed that the search algorithm chosen by the choosing procedure does not return to any points in $d_\cup$.[5]

The following theorem, proven in Appendix G, establishes that there is no *a priori* justification for using any particular choosing procedure. Loosely speaking, no matter what the cost function, without special consideration of the algorithm at hand, simply observing how well that algorithm has done so far tells us nothing *a priori* about how well it would do if we continue to use it on the same cost function. For simplicity, in stating the result we only consider deterministic algorithms.

Theorem 9 *Let d_m and d'_m be two fixed samples of size m, that are generated when the algorithms a and a' respectively are run on the (arbitrary) cost function at hand. Let A and B be two different choosing procedures. Let k be the number of elements in $c_{>m}$. Then*

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

Implicit in this result is the assumption that the sum excludes those algorithms a and a' that do not result in d and d' respectively when run on f.

In the precise form it is presented above, the result may appear misleading, since it treats all populations equally, when for any given f some populations will be more likely 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 result, proven in Appendix H:

Theorem 10 *Under the conditions given in the preceding theorem,*

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

These results show that no assumption for $P(f)$ alone 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 is choosing among. This conclusion may be surprising. In particular, note that it means that there is no intrinsic advantage to using a rational choosing procedure, which continues with the better of a and a', rather than using a irrational choosing procedure which does the opposite.

These results also have interesting implications for degenerate choosing procedures $A \equiv \{$always use algorithm $a\}$, and $B \equiv \{$always use algorithm $a'\}$. As applied to this case, they

---

[5]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 convention we exploited 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 $d_{>m}$.

mean that for fixed f_1 and f_2, if f_1 does better (on average) with the algorithms in some set $\mathcal{A}$, then f_2 does better (on average) with the algorithms in the set of all other algorithms. In particular, if for some favorite algorithms a certain "well-behaved" f results in better performance than does the random f, then that well-behaved f gives *worse than random* behavior on the set all remaining algorithms. In this sense, just as there are no universally efficacious search algorithms, there are no universally benign f which can be assured of resulting in better than random performance regardless of one's algorithm.

In fact, things may very well be worse than this. In supervised learning, there is a related result [Wol96a]. Translated into the current context that result suggests that if one restricts our sums to only be over those algorithms that are a good match to $P(f)$, then it is often the case that "stupid" choosing procedures — like the irrational procedure of choosing the algorithm with the less desirable $\vec{c}$ — outperform "intelligent" ones. What the set of algorithms summed over must be for a rational choosing procedure to be superior to an irrational is not currently known.

8   Conclusions

A framework has been presented in which to compare general-purpose optimization algorithms. A number of NFL theorems were derived that demonstrate the danger of comparing algorithms by their performance on a small sample of problems. These same results also indicate the importance of incorporating problem-specific knowledge into the behavior of the algorithm. A geometric interpretation was given showing what it means for an algorithm to be well-suited to solving a certain class of problems. The geometric perspective also suggests a number of measures to compare the similarity of various optimization algorithms.

More direct calculational applications of the NFL theorem were demonstrated by investigating certain information theoretic aspects of search, as well as by developing a number of benchmark measures of algorithm performance. These benchmark measures should prove useful in practice.

We provided an analysis of the ways that algorithms can differ *a priori* despite the NFL theorems. We have also provided an introduction to a variant of the framework that focuses on the behavior of a range of algorithms on specific problems (rather than specific algorithms over a range of problems). This variant leads directly to reconsideration of many issues addressed by computational complexity, as detailed in [MW96].

Much future work clearly remains — the reader is directed to [WM95] for a list of some of it. Most important is the development of practical applications of these ideas. Can the geometric viewpoint be used to construct new optimization techniques in practice? We believe the answer to be yes. At a minimum, as Markov random field models of landscapes become more wide-spread, the approach embodied in this paper should find wider applicability.

Inc. for support.

References

[CT91]   T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, New York, 1991.

[FOW66] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, New York, 1966.

[Glo89]   F. Glover. *ORSA J. Comput.*, 1:190, 1989.

[Glo90]   F. Glover. *ORSA J. Comput.*, 2:4, 1990.

[Gri76]   D. Griffeath. Introduction to random fields. Springer-Verlag, New York, 1976.

[Hol93]   J. H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, 1993.

[KGV83] S. Kirkpatrick, D. C. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671, 1983.

[KS80]   R. Kinderman and J. L. Snell. *Markov random fields and their applications*. American Mathematical Society, Providence, 1980.

[LW66]   E. L. Lawler and D. E. Wood. *Operations Research*, 14:699–719, 1966.

[MW96]   W. G. Macready and D. H. Wolpert. What makes an optimization problem? *Complexity*, 5:40–46, 1996.

[WM95]   D. H. Wolpert and W. G. Macready. No free lunch theorems for search. Technical Report SFI-TR-05-010; ftp : $//ftp.santafe.edu/pub/dhw_ftp/nfl.search.TR.ps.Z$, Santa Fe Institute, 1995.

[Wol96a]   D. H. Wolpert. The lack of a prior distinctions between learning algorithms and the existence of a priori distinctions between learning algorithms. *Neural Computation*, 1996.

[Wol96b] D. H. Wolpert. On bias plus variance. Neural Computation, in press, 1996.

A   NFL proof for static cost functions

We show that $\sum_f P(\vec{c} \mid f, m, a)$ has no dependence on a. Conceptually, the proof is quite simple but necessary book-keeping complicates things, lengthening the proof considerably. The intuition behind the proof is quite simple though: by summing over all f we ensure that

the past performance of an algorithm has no bearing on its future performance. Accordingly, under such a sum, all algorithms perform equally.

The proof is by induction. The induction is based on $m = 1$ and the inductive step is based on breaking f 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.

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

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

where δ is the Kronecker delta function.

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

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

which is independent of a. This bases the induction.

The inductive step requires that if $\sum_f P(d_m^y | f, m, a)$ is independent of a for all d_m^y, then so also is $\sum_f P(d_{m+1}^y | f, m + 1, a)$. Establishing this step completes the proof.

We begin by writing

$$P(d_{m+1}^y | f, m + 1, a) = P(\{d_{m+1}^y(1), \ldots, d_{m+1}^y(m)\}, d_{m+1}^y(m + 1) | f, m + 1, a)$$
$$= P(d_m^y, d_{m+1}^y(m + 1) | f, m + 1, a)$$
$$= P(d_{m+1}^y(m + 1) | d_m, f, m + 1, a) \, P(d_m^y | f, m + 1, a)$$

and thus

$$\sum_f P(d_{m+1}^y | f, m + 1, a) = \sum_f P(d_{m+1}^y(m + 1) | d_m^y, f, m + 1, a) \, P(d_m^y | f, m + 1, a).$$

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

$$\sum_f P(d_{m+1}^y | f, m+1, a) = \sum_{f,x} P(d_{m+1}^y(m + 1) | f, x) \, P(x | d_m^y, f, m+1, a) P(d_m^y | f, m + 1, a)$$
$$= \sum_{f,x} \delta(d_{m+1}^y(m + 1), f(x)) \, P(x | d_m^y, f, m+1, a) P(d_m^y | f, m + 1, a).$$

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

$$\sum_f P(d_{m+1}^y | f, m+1, a) = \sum_{f,x,d_m^x} \delta(d_{m+1}^y(m + 1), f(x)) \, P(x | d_m, a) \, P(d_m^x | d_m^y, f, m + 1, a)$$
$$\times P(d_m^y | f, m + 1, a)$$
$$= \sum_{f, d_m^x} \delta(d_{m+1}^y(m + 1), f(a(d_m))) \times P(d_m | f, m, a)$$

23

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

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

$$\sum_f P(d_{m+1}^y|f, m+1, a) = \sum_{d_m^x} \sum_{f(x \in d_m^x)} P(d_m|f, m, a) \sum_{f(x \notin d_m^x)} \delta(d_{m+1}^y(m+1), f(a(d_m))). \qquad (8)$$

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

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

$$= \frac{1}{|\mathcal{Y}|} \sum_{f, d_m^x} P(d_m|f, m, a)$$

$$= \frac{1}{|\mathcal{Y}|} \sum_f P(d_m^y|f, m, a)$$

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.

B   NFL proof for time-dependent cost functions

In analogy with the proof of the static NFL theorem, the proof for the time-dependent case proceeds by establishing the a-independence of the sum $\sum_T P(c| f, T, m, a)$, where here c is either d_m^y or D_m^y.

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

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

$$= \sum_{d_m^x} \sum_{f_2 \cdots f_m} P(\vec{c} \mid \vec{f}, d_m^x) P(d_m^x \mid \vec{f}, m, a) \sum_T P(f_2 \cdots f_m \mid f_1, T, m, a),$$

where the sequence of cost functions, f_i, has been indicated by the vector $\vec{f} = (f_1, \cdots, f_m)$. In the next step, the sum over all possible T is decomposed 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

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

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

24

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

Consider the innermost sum over T_{m-1}, for fixed values of the outer sum indices $T_1 \ldots T_{m-2}$. For fixed values of the outer indices $T_{m-1}(T_{m-2}(\cdots T_1(f_1)))$ is just a particular 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 the constant, $(|\mathcal{F}|-1)!$. Consequently, evaluating the T_{m-1} sum yields

$$\sum_T P(c|f, T, m, a_1) \propto \sum_{d_m^x} \sum_{f_2 \cdots f_m} P(c|\vec{f}, d_m^x) P(d_m^x \mid \vec{f}, m, a)$$
$$\times \sum_{T_1} \delta(f_2, T_1(f_1)) \cdots \sum_{T_{m-2}} \delta(f_{m-1}, T_{m-2}(T_{m-3}(\cdots T_1(f_1)))).$$

The sum over T_{m-2} can be accomplished in the same manner T_{m-1} is summed over. In fact, all the sums over all T_i can be done, leaving

$$\sum_T P(c|f, T, m, a_1) \propto \sum_{d_m^x} \sum_{f_2 \cdots f_m} P(D_m^y|\vec{f}, d_m^x) P(d_m^x \mid \vec{f}, m, a)$$
$$= \sum_{d_m^x} \sum_{f_2 \cdots f_m} P(c|\vec{f}, d_m^x) P(d_m^x \mid f_1 \cdots f_{m-1}, m, a). \tag{9}$$

In this last step the statistical independence of c and f_m has been used.

Further progress depends on whether c represents d_m^y or D_m^y. We begin with analysis of the D_m^y case. For this case $P(c|\vec{f}, d_m^x) = P(D_m^y|f_m, d_m^x)$, since D_m^y only reflects cost values from the last cost function, f_m. Using this result gives

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

The final sum over f_m is a constant equal to the number of ways of generating the sample D_m^y from cost values drawn from f_m. The important point is that it is independent of the particular d_m^x. Because of this the sum over d_m^x can be evaluated eliminating the a dependence.

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

This completes the proof of Theorem 2 for the case of D_m^y.

The proof of Theorem 2 is completed by turning to the d_m^y case. This is considerably more difficult since $P(\vec{c} \mid \vec{f}, d_m^x)$ can not be simplified so that the sums over f_i can not be decoupled. Nevertheless, the NFL result still holds. This is proven by expanding Equation (9) over possible d_m^y values.

$$\sum_T P(d_m^y|f, T, m, a) \propto \sum_{d_m^x} \sum_{f_2 \cdots f_m} \sum_{d_m^y} P(d_m^y|d_m^y) P(d_m^y \mid \vec{f}, d_m^x) P(d_m^x \mid f_1 \cdots f_{m-1}, m, a)$$
$$= \sum_{d_m^y} P(d_m^y|d_m^y) \sum_{d_m^x} \sum_{f_2 \cdots f_m} P(d_m^x \mid f_1 \cdots f_{m-1}, m, a) \prod_{i=1}^m \delta(d_m^y(i), f_i(d_m^x(i))) \tag{10}$$

The innermost sum over 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 $|\mathcal{Y}|^{|\mathcal{X}|-1}$. This leaves

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

The sum over $d_m^x(m)$ is now simple,

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

The above equation is of the same form as Equation (10), only with a remaining population of size $m-1$ rather than m. Consequently, in an analogous manner to the scheme used to evaluate the sums over f_m and $d_m^x(m)$ that existed in Equation (10), the sums over f_{m-1} and $d_m^x(m-1)$ can be evaluated. Doing so simply generates more a-independent proportionality constants. Continuing in this manner, all sums over the f_i can be evaluated to find

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

There is algorithm-dependence in this result but it is the trivial dependence discussed previously. It arises from how the algorithm selects the first x point in its population, $d_m^x(1)$. Restricting interest to those points in the sample that are generated subsequent to the first, this result shows that there are no distinctions between algorithms. Alternatively, summing over the initial cost function f_1, all points in the sample could be considered while still retaining an NFL result.

C   Proof of ρ_f result

As noted in the discussion leading up to Theorem 3 the fraction of functions giving a specified histogram $\vec{c} = m\vec{\alpha}$ is independent of the algorithm. Consequently, a simple algorithm is used to prove the theorem. The algorithm 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. The number of f's giving the desired histogram under this 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 giving the first result of Theorem 3.

The expression of $\rho_f(\vec{\alpha})$ in terms of the entropy of $\vec{\alpha}$ follows from an application of Stirling's approximation to order $\mathcal{O}(1/m)$, which is valid when all of the c_i are large. In this

case the multinomial is written:

$$\ln \binom{m}{c_1\, c_2\, \cdots\, c_{|\mathcal{Y}|}} \cong m \ln m - \sum_{i=1}^{|\mathcal{Y}|} c_i \ln c_i + \frac{1}{2}\left[\ln m - \sum_{i=1}^{|\mathcal{Y}|} \ln c_i\right]$$

$$\cong m\, S(\vec{\alpha}) + \frac{1}{2}\left[\left(1 - |\mathcal{Y}|\right)\ln m - \sum_{i=1}^{|\mathcal{Y}|} \ln \alpha_i\right]$$

from which the theorem follows by exponentiating this result.

D   Proof of ρ_{alg} result

In this section the proportion of all algorithms that give a particular $\vec{c}$ for a particular f is calculated. The calculation proceeds in several steps:

Since $\mathcal{X}$ is finite there are finite number of different samples. Therefore any (deterministic) a is a huge – but finite – list indexed by all possible d's. Each entry in the list is the x the a in question outputs for that d-index.

Consider any particular unordered set of m $(\mathcal{X}, \mathcal{Y})$ pairs where no two of the pairs share the same x value. Such a set is called 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 Equation (3) is the number of unordered paths in the given f that give the desired $\vec{c}$.

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

Proof: The proof is established by constructing a mapping $\phi : a \mapsto \pi$ taking in an a that gives the desired $\vec{c}$ for f, and producing a π that is in f and gives the desired $\vec{c}$. Showing that for any π the number of algorithms a such that $\phi(a) = \pi$ is a constant, independent of π, f, and $\vec{c}$. and that ϕ is single-valued will complete the proof.

Recalling that that every x value in an unordered path is distinct any unordered path π gives a set of $m!$ different ordered paths. Each such ordered path π_{ord} in turn provides a set of m successive d's (if the empty d is included) and a following x. Indicate by $d(\pi_{ord})$ this set of the first m d's provided by π_{ord}.

¿From any ordered path π_{ord} a "partial algorithm" can be constructed. 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. Since there are $m!$ distinct partial a's for each π (one for each ordered path corresponding to π), there are $m!$ such partially filled-in lists for each π. A partial algorithm may or may not be consistent with a particular full algorithm. This allows the definition of 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).

To complete the first part of the proof it must be shown 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. Now 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. 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.

Reordering is accomplished by interchanging pairs of d indices. First, interchange any d index of the form $((d_m^x(1), d_m^y(1)), \dots, (d_m^x(i \leq m), d_m^y(i \leq m)))$ whose entry is filled in in any of our partial algorithm lists with $d'(d) \equiv ((d_m^x(1), z), \dots, (d_m^x(i), z))$, 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 $x \in \mathcal{X}$: $(x_1, \dots, x_{|\mathcal{X}|})$. Then interchange any d' index of the form $((d_m^x(1), z, \dots, (d_m^x(i \leq m), z)$ whose entry is filled in in any of our (new) partial algorithm lists with $d''(d') \equiv ((x_1, z), \dots, (x_m, z))$. Recall that all the $d_m^x(i)$ must be distinct. 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.

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. 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.

To show the relation to the Kullback-Liebler distance the product of binomials is expanded with the aid of Stirlings approximation when both N_i and c_i are large:

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

We it has been assumed that $c_i / N_i \ll 1$, which is reasonable when $m \ll |\mathcal{X}|$. Expanding $\ln(1-z) = -z - z^2/2 - \cdots$, to second order gives

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

28

Using $m/|\mathcal{X}| \ll 1$ then in terms of $\vec{\alpha}$ and $\vec{\beta}$ one finds

$$\ln \prod_{i=1}^{|\mathcal{Y}|} \binom{N_i}{c_i} \cong -m D_{KL}(\vec{\alpha}, \vec{\beta}) + m - m \ln\left(\frac{m}{|\mathcal{X}|}\right) - \frac{|\mathcal{Y}|}{2} \ln 2\pi$$

$$- \sum_{i=1}^{|\mathcal{Y}|} \frac{1}{2} \ln(\alpha_i m) + \frac{m}{2|\mathcal{X}|}\left(\frac{\alpha_i}{\beta_i}\right)(1 - \alpha_i m + \cdots),$$

where $D_{KL}(\vec{\alpha}, \vec{\beta}) \equiv \sum_i \alpha_i \ln(\beta_i/\alpha_i)$ is the Kullback-Liebler distance between the distributions $\vec{\alpha}$ and $\vec{\beta}$. Exponentiating this expression yields the second result in Theorem 4.

E Benchmark measures of performance

The result for each benchmark measure is established in turn.

The first measure is $\sum_f P(\min(d_m^y)|f, m, a)$. Consider

$$\sum_f P(\min(d_m^y)|f, m, a) \qquad (11)$$

for which the summand equals 0 or 1 for all f and deterministic a. It is 1 only if

  i) $f(d_m^x(1)) = d_m^y(1)$

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

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

and so on. These restrictions will fix the value of $f(x)$ at m points while f remains free at all other points. Therefore

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

Using this result in Equation (11) we find

$$\sum_f P(\min(d_m^y) > \epsilon \mid f, m) = \frac{1}{|\mathcal{Y}|^m} \sum_{d_m^y} P(\min(d_m^y) > \epsilon \mid d_m^y) = \frac{1}{|\mathcal{Y}|^m} \sum_{d_m^y \ni \min(d_m^y) > \epsilon} 1$$

$$= \frac{1}{|\mathcal{Y}|^m}(|\mathcal{Y}| - \epsilon)^m.$$

which is the result quoted in Theorem 5.

In the limit as $|\mathcal{Y}|$ gets large write $\sum_f E(\min(\vec{c})|f, m) = \sum_{\epsilon=1}^{|\mathcal{Y}|} \epsilon[\omega^m(\epsilon-1) - \omega^m(\epsilon)]$ and substitute in for $\omega(\epsilon) = 1 - \epsilon/|\mathcal{Y}|$. Replacing ϵ with $\zeta + 1$ turns the sum into $\sum_{\zeta=0}^{|\mathcal{Y}|-1}[\zeta + 1] [(1 - \frac{\zeta}{|\mathcal{Y}|})^m - (1 - \frac{\zeta+1}{|\mathcal{Y}|})^m]$. Next, write $|\mathcal{Y}| = b/\Delta$ for some b and multiply and divide the summand by Δ. Since $|\mathcal{Y}| \to \infty$ then $\Delta \to 0$. To take the limit of $\Delta \to 0$, apply L'hopital's

rule to the ratio in the summand. Next use the fact that Δ is going to 0 to cancel terms in the summand. Carrying through the algebra, and dividing by b/Δ, we get a Riemann sum of the form $\frac{m}{b^2} \int_0^b dx \; x(1 - x/b)^{m-1}$. Evaluating the integral gives the second result in Theorem 5.

The second benchmark concerns the behavior of the random algorithm. Marginalizing over the $\mathcal{Y}$ values of different 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 $\prod_{i=1}^{|\mathcal{Y}|} \binom{N_i}{c_i} / \binom{|\mathcal{X}|}{m}$. So

$$P(\min(\vec{c}) \geq \epsilon \,|\, f, m, \tilde{a}) = \frac{1}{\binom{|\mathcal{X}|}{m}} \sum_{c_1=0}^{m} \cdots \sum_{c_{|\mathcal{Y}|}=0}^{m} \delta(\sum_{i=1}^{|\mathcal{Y}|} c_i, m) P(\min(\vec{c}) \geq \epsilon|\vec{c}) \prod_{i=1}^{|\mathcal{Y}|} \binom{N_i}{c_i}$$

$$= \frac{1}{\binom{|\mathcal{X}|}{m}} \sum_{c_\epsilon=0}^{m} \cdots \sum_{c_{|\mathcal{Y}|}=0}^{m} \delta(\sum_{i=\epsilon}^{|\mathcal{Y}|} c_i, m) \prod_{i=\epsilon}^{|\mathcal{Y}|} \binom{N_i}{c_i}$$

$$= \frac{\binom{\sum_{i=\epsilon}^{|\mathcal{Y}|} N_i}{m}}{\binom{|\mathcal{X}|}{m}} \equiv \frac{\binom{\Omega(\epsilon)|\mathcal{X}|}{m}}{\binom{|\mathcal{X}|}{m}}$$

which is Equation (4) of Theorem 6.

F Proof related to minimax distinctions between algorithms

The proof is 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:

30

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 first 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.

G   Fixed cost functions and choosing procedures

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

$$\sum_{a, a'} P(c_{>m} \mid f, d_1, d_2, k, a, a', proc) = \sum_{a_{\perp d}, a'_{\perp d'}} \sum_{a_{\subset d}, a'_{\subset d'}} \sum_{a_{\supseteq d}, a'_{\supseteq d'}} P(c_{>m} \mid f, d, d', k, a, a', proc).$$

31

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 $proc$, the 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 the 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 the 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, the sum can be rewritten 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 $proc$.

H   Proof of Theorem 9

Let $proc$ refer to a choosing procedure. We are interested in

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

The sum over d and d' can be moved 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 the assumption 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.