University of Alberta

On the Futility of Blind Search

by

Joseph C. Culberson

Technical Report TR 96-18
July 1996

DEPARTMENT OF COMPUTING SCIENCE
The University of Alberta
Edmonton, Alberta, Canada

On the Futility of Blind Search

Joseph C. Culberson *

September 5, 1996

Abstract

This paper might have been subtitled "An algorithmicist looks at no free lunch."

We use simple adversary arguments to redevelop and explore some of the no free lunch (NFL) theorems and perhaps extend them a little. A second goal is to clarify the relationship of NFL theorems to algorithm theory. In particular we claim that NFL puts much weaker restrictions on the claims that an evolutionary algorithm can make than does acceptance of the conjectures of traditional complexity theory. And third we take a brief look at whether the notion of natural evolution relates to optimization, and what if any the implications of evolution are for computing. In this part, we mostly try to raise questions concerning the validity of applying the genetic model to the problem of optimization.

This is an informal paper — most of the information presented is not formally proven, and is either "common knowledge" or formally proven elsewhere. Some of the claims are intuitions based on experience with algorithms, and in a more formal setting should be classified as conjectures. The goal is not so much to develop theory, as it is to perhaps persuade the reader to adopt a particular viewpoint.

1 Introduction

In the movie *UHF*, there is a marvelous scene that every computing scientist should consider. As the camera slowly pans across a small park setting, we hear a voice repeatedly asking "Is this it?" followed each time by the

*Supported by Natural Sciences and Engineering Research Council Grant No. OGP8053. Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, T6G 2H1. email:joe@cs.ualberta.ca

response "Nah!". As the camera continues to pan, it picks up two men on a park bench, one of them blind and holding a Rubik's cube. He randomly gives it a twist, then holds it up to his friend to repeat the question/answer sequence yet again.

This is blind search. The searcher has a minimum of information, and gets back a minimal amount of response in terms of information from the environment. In the literature on diverse areas related to search, there are numerous papers purporting to address the issue of a general problem solver, no knowledge search engine, universal learning machines, arbitrary representation evolvers, or other blind search mechanisms. In general, there is an understandable desire to have a "magic bullet" algorithm that can solve any problem. Of course, the results of Turing [23] already forbid this, but still there seems to be a persistent belief that it should be possible to have an algorithm that outperforms all others, or that we can somehow identify an operator that will work to our advantage under any circumstances.

The No Free Lunch (NFL) theorems of Wolpert and Macready [24] point out the inherent hopelessness of such beliefs. In fairness of course, few researchers would be prepared to defend the *complete generality* of their proposals if pressed. Still, these theorems have generated much controversy throughout the Evolutionary Algorithms (EA) community in part because not everyone seems willing to accept the full implications.

Although most researchers would readily accept that there must be some regularity inherent in a function if an algorithm is going to make progress, Wolpert and Macready make the further point that unless the operators of the algorithm are correlated to those features, there is no reason to believe the algorithm will do better than a random search. Yet when such operators are evaluated, the description of the conditions under which the particular operator or algorithm will succeed or fail is often slighted. For example, there are numerous papers in conferences from the early years of GAs purporting to show that two point crossover is better than one point, or that crossover is more (or less) effective than mutation or that certain selection methods outperform others, and so forth. These claims were often supported by some type of global analysis (frequently based on the Schema Theorem) and selected experiments on various test suites. The NFL theorems make it clear that any such claim is inherently false unless it is coupled with a disclaimer such as "under the prescribed assumptions" or "on the functions tested." In particular, no such claim can ever be made with respect to all possible functions, or even sufficiently large classes of functions as we illustrate below.

There has been much debate [21] on the interpretation and meaning of the no free lunch (NFL) theorems for search. Some of this has centered on whether they are applicable to "real life" problems, some on the relationship to standard complexity theory. This paper will it is hoped, serve three purposes. First, it will present the NFL results using simple informal adversary arguments that may be more intuitive, at least for those from an algorithms background.

Second, we will attempt to make convincing, although informal arguments to relate NFL to traditional complexity theory. Our view is that the restrictions imposed by NFL are of much weaker impact than the claims of algorithms theory based on conjectures such as that all the following are proper inclusions

$$\mathrm{P} \subset \mathrm{RP} \subset \mathrm{NP} \subset \mathrm{PSPACE}$$

Finally, we present a short section on Myopic Search and the idea of competitive edges; that is, search where we have partial information, but not complete knowledge of the problem, and would be happy to make some gain over known solutions. In some classes, the NFL theorems do not apply, at least not in their full generality. We contend that it is here that research on adaptive algorithms, incorporating as much knowledge as is available, is most likely to be fruitful. We also speculate on the nature of natural genetics from a computational search perspective, and ponder the somewhat philosophical question of how life could arise despite the NFL theorems. In particular, we contend that the fact that life evolved does not imply that genetic search is an efficient *universal* problem solver.

Part I
An Adversary Approach to NFL

2 Computational Models and Blind Search

Genetic algorithms (GAs) are often touted as "no prior knowledge" algorithms. This means that we expect GAs to perform without special information from the environment. Similar claims are often made for other adaptive and evolutionary algorithms.

One computational model of this expectation is the following. An algorithm A generates a binary string which is then evaluated by the environment $\mathcal{E}$. This is the only means the algorithm has of communicating with the environment. The environment acts as a black box, and so we refer to this as the *black box* or *blind search* model. I suppose we could also call it the "friend on a park bench model". Since almost all computers in current use are based on binary logic, the restriction to a domain of binary strings will be enforced in practice as soon as we try to implement any optimization algorithm, and so is not a significant restriction. For example, the analysis also applies to domains using larger alphabets, since larger alphabets can and must be encoded using binary strings. The observations we make will apply to any search algorithm, such as genetic algorithms, simulated annealing, neural nets, hill climbers and many, many others.

The question arises, "is this a reasonable model to pit an algorithm against?" Wolpert and Macready's "no free lunch" theorem [24] answers this question in the negative. They prove within a formal setting that *all* optimization algorithms have equivalent average behavior when pitted against such a black box environment.

In this presentation, let us use a different approach to this theorem based on a standard tool of complexity analysis, the *adversary argument.* Adversaries are ideally suited to the black box model. We assume that an adversary is sitting in the black box, and that each time an input is to be evaluated the adversary chooses the value of the string. We may place various constraints and conditions on the values the adversary can choose to mimic various computational models we have in mind. However, the important idea of an adversary is that it gets to choose its values at the time the request is made; that is, the adversary constructs the function being searched "on demand."

For our version of the NFL, we consider the classes of functions $\mathcal{F}_n = \{f : \mathrm{S}^n \to \mathcal{R}\}$, where $\mathcal{R}$ is some finite discrete range and $\mathrm{S} = \{0, 1\}$. We will normally drop the subscript n. For this paper we will usually assume that $\mathcal{R}$ has a total ordering, and that the objective of the search is to find some string in S that is optimal under f. In order to ensure that the functions are defined over all of S^n, we may insist in our model that the algorithm will always eventually generate every binary string in the domain. See [24] for a discussion of this restriction. The arguments below do not really depend on this restriction. We use $N = 2^n$ for the number of different strings in S^n.

In order to compare algorithms, we need a measure of performance for any algorithm A on a function f, denoted $\mathbf{M}_{\mathrm{A}}(f)$. We extend our notation with $\mathbf{M}_{\mathrm{A}}(\mathbf{F})$ to mean the average performance over all functions in the class $\mathbf{F}$. In practice this measure may be the (expected) number of string evaluations until an optimum string is generated, or some measure of goodness of an approximation after a specified number of evaluations. Wolpert and Macready consider the histogram of values seen after m distinct strings are evaluated. The above measures are then derivable from the histogram. The NFL states that the histogram, and thus any measure of performance based on it is independent of the algorithm if all functions are considered equally likely.

Observation: NFL(1): For all algorithms $\mathrm{A}_1, \mathrm{A}_2$**,** $\mathbf{M}_{\mathrm{A}_1}(\mathcal{F}) = \mathbf{M}_{\mathrm{A}_2}(\mathcal{F})$ **in the black box model.**

Proof: Consider the following adversary. When $\mathcal{E}$ receives a string, if the string has been previously evaluated, the adversary returns the previously used value, otherwise it assigns a randomly generated value from $\mathcal{R}$. Thus, the sequence (histogram) of values observed is independent of the sequence of generated strings, and so expectations are equal.

For any algorithm, the adversary process randomly generates a function with equal probability from $\mathcal{F}$ (or a partial function if the algorithms do not generate all possible strings and we only count up to the minimum number generated), since the string-value pairs of the function are independently and uniformly assigned. ■

We let RE represent a random enumeration search; a search which generates n-bit strings in random order without repetition. Although this algorithm is not practical to implement, it serves as a theoretical basis of comparison. A practical algorithm, in the sense that it is easy to implement even if not particularly efficient on any algorithm, is a cyclic (mod N) search starting from a random initial point. This cyclic algorithm has the property that it uses the same expected number of string evaluations to find the optimum as RE against *any* adversary, even if the adversary is restricted to a single function.

An immediate corollary of the NFL is that all algorithms are equal to RE under the blind search model of computation.

We can refine the NFL theorem by noting that even if we constrain the adversary to make its selections from $\mathcal{R}$ without repetition the proof still holds. Thus, even if we restrict $\mathcal{F}_n$ to bijections and $\mathcal{R} = [1, 2, \ldots, 2^n]$, we can do no better than a random enumeration on average.

In practice we may wish to restrict our attention to some subclass of functions $\mathbf{F} \subset \mathcal{F}$. The following is immediate:

Observation: NFL(2): If $\mathbf{M}_{\mathrm{A}}(\mathbf{F}) < \mathbf{M}_{\mathrm{RE}}(\mathbf{F})$ **then** $\mathbf{M}_{\mathrm{A}}(\mathcal{F} \setminus \mathbf{F}) > \mathbf{M}_{\mathrm{RE}}(\mathcal{F} \setminus \mathbf{F})$**.**

Thus any algorithm which attempts to exploit some property of functions in a subclass of $\mathcal{F}$ will be subject to deception for some other class, in the sense that it will perform worse than random search. The situation is even more bleak than this might seem to imply. Informally, let $\mathbf{F}_{\mathrm{A}}(\xi)$ be the class of functions for which the optimal string is found (i.e. is in the histogram) "on average" after (at most) ξ string evaluations using algorithm A.

Observation: The proportion of bijections solved by any A with expected efficiency ξ is

$$|\mathbf{F}_{\mathrm{A}}(\xi)| \leq \frac{\xi}{2^n}$$

Proof: (Informal) We use the same adversary as before. Since the value sequence is independent of the string generation order, we need only note that the claim is based simply on the fraction of runs in which the adversary generates the optimal value within the first ξ strings. ■

This says for example that any algorithm that we wish to run in polynomial (i.e. in n^k) expected time can do so on only an exponentially small ($n^k/2^n$) fraction of the set of all possible functions.

In summary, the basic problem is that to apply any algorithm to *all possible functions* (or even all possible bijections on a restricted range) we have to give the adversary far too much power. As a result, the adversary is able to generate values for us that are completely independent of the algorithm in use.

This power carries over even if we are only interested in marginal improvement, for example in a competitive random environment between two or more algorithms. Since the adversaries generate the next values completely at random, it is irrelevant what method the algorithms use to generate their next strings. Who wins the next round is determined by the adversary independently of the algorithms, provided they both generate strings not previously seen.

3 The Futility of Blind Transformations of the Encoding

Many papers discuss isomorphisms and transformations of functions and search spaces, sometimes directly as for example in the use of Gray codings, and sometimes in terms of alternative encodings.

Observation: If $\mathcal{H} : \mathbf{F} \to \mathbf{F}$ is a bijection then for any A, $\mathrm{M_A}(\mathcal{H}(\mathbf{F})) = \mathrm{M_A}(\mathbf{F})$.

We can conclude, for example as noted in [3], that applying Gray codes to all the functions in $\mathcal{F}$ yields no improvement for any algorithm on average over $\mathcal{F}$. In fact, the notion is considerably tighter than that, because it applies to any class $\mathbf{F}$ closed under $\mathcal{H}$. For example, suppose we start with some particular function, say one of the De Jong suite functions [14] and then apply Gray codes to this function to obtain a new function. We may then apply Gray codes to that function and so on, until eventually we obtain the original De Jong function again. This must occur since Gray coding induces a permutation on S^n. As shown in [9], it will take $2^{\lceil \log_2 n \rceil} < 2n$ iterations to return to the initial function. On this set of less than $2n$ functions, the average performance of any algorithm is unaffected by the application of Gray coding.

It might be argued that for *interesting* functions, Gray coding will help a GA. If we define *interesting* to mean a function for which Gray coding helps a GA, then this statement is trivially true but pointless! Let us use a more reasonable definition, one based on complexity, such as f is interesting if it requires at most $O(n^2)$ work to evaluate any string. Then if $\mathbf{F}$ is the set of functions generated by iterated Gray coding of any interesting function f, by definition all functions in $\mathbf{F}$ are interesting because the iterated Gray code can be applied in $O(n^2)$ time. Yet applying Gray coding to this class does not on average have any effect, for a GA or any other algorithm.

One implication of these observations is that the blind application of transformations of the encoding has no expectation of improved performance. Another is that attempts to have the GA evolve encodings in order to improve its performance are generally fruitless, unless there is some property of the function class we are observing that correlates to the evolutionary mechanism and we can show that this property is exploited by our algorithm. This comment merits particular attention when we consider natural evolution, and the possibility that nature has evolved an effective search engine in the form of bisexual exchange of DNA (although we seriously question the notion of function optimization in this context).

4 Free Lunch Analysis: The Schema Theorem

Holland's Schema Theorem [18] at one time was seen as the guiding light of GA analysis and design, but now schema-bashing seems to be an accepted art form in the GA community (see the references for example in the introduction in [2]). The schema theorem, on binary strings, identifies subsets of the binary strings which correspond to hyperplanes of the Hamming cube (or equivalently similarity templates) as being important to canonical GA performance. Roughly it says that, allowing for disruption and recreation of schema due to the mutation and recombination operators, if we have an infinite population then after each generation of the GA the proportion of strings belonging to any such subset will increase in proportion to the ratio of the average fitness of the subset relative to the population as a whole.

So why is this an attempt at a "Free Lunch"? Well, as a theorem it is not; but if we try to apply it to the design of GAs or other algorithms in the absence of other information about the function to be searched then it is subject to failure for exactly the reasons outlined in the NFL theorems. The problem is that decoupling the theorem from its assumptions, or from the particulars of a specific algorithm, or from the particulars of a particular function, *all* allow an adversary to falsify the predictions of the analysis.

In order to gain some insight consider the numerous other theorems related to other subset systems over S^n that are all equally valid. A *cover* of a set S is a collection of subsets whose union includes every element in the set. The number of covers of a set of N elements is given by the recurrence [8]

$$C_N = \begin{cases} 2^{2^N-1} - \sum_{i=0}^{N-1} \binom{N}{i} c_i, & i > 0 \\ 1, & i = 0 \end{cases}$$

Since for binary strings of length n the number of distinct strings is $N = 2^n$, it follows that the number of covers grows approximately as

$$C_{\mathrm{S}} \approx 2^{2^{2^n}}$$

For every cover there is a theorem similar to the schema theorem when proscribed with the appropriate disruption and recombination probabilities. Thus, the schema theorem is only one out of these super-exponentially many equally valid theorems. Admittedly, some of the covers, such as the one consisting of the single universal set, are rather trivial. But each theorem being equally valid under its assumptions lays equal claim as a basis of analysis and design for a proportional selection algorithm.

These theorems are mostly about the processes of selection and reproduction and yield only peripheral information about the effect of combinatorial operators. These theorems are really only side effects of the fact that higher valued strings are more likely to reproduce under this particular selection mode. They do not tell us much about the design of search algorithms using those operators and especially they do not tell us whether selection based on relative fitness is a good idea for an arbitrary function, or a good idea in conjunction with a particular set of combinatorial operators. As Wolpert and Macready [24] point out, an algorithm that moves downhill is just as good on average as one that moves uphill. Trying to apply any of these theorems as a design tool in the absence of specific information is trying to get a free lunch.

Part II
Complexity Theory and NFL

In their two related papers [24, 20], and in particular in section 5 of the latter, Wolpert and Macready discuss the relationship of their NFL results to standard complexity. However, their discussion is somewhat misleading and contains some inaccuracies.

In this part of the paper we present an alternative viewpoint.

5 Laws of the Computational Universe

In physics there is the law of conservation of matter and energy, in chemistry there are the laws of thermodynamics and in computing science there is a similar law known as Church's thesis based on an observation made by Alonzo Church in 1936(See [6] or [22] for an exposition). In today's terminology, this "law" says roughly that any algorithm that can be specified by a human and carried out on any conceivable realistic computer can also be programmed on a Turing machine (TM). Note that there is no way to prove such a claim, just as there is no way to prove that physical law applies throughout the universe. But all sufficiently powerful formal models abstracting the notion of what it means to do computation have been shown to be formally equivalent to TMs in the strong sense that each can emulate all the others.

Turing machines occupy a role in computation similar to the roles within the physical sciences occupied by ideal gases, black box radiators and the use of point mass representations in the computation of planetary orbits. A TM is an abstraction of the fundamental qualities of any computational process. A TM may not be physically realizable, in the sense that we have no way of providing *infinite* memory, but still any limitation resulting from the inherent properties of TM's represents at least the same limitation on real computers. For example, there are many problems including the infamous Halting problem which can be proven to be unsolvable on a TM and thus by implication of Church's thesis on any conceivable computational device.

A TM is a mathematical model of computation which consists of a finite state machine, or intuitively a program, and an infinite read/write tape device. At each time step the program reads one input character from the tape, changes state, writes out one symbol to the tape, and moves the tape read/write head one step right or left. Initially the machine starts in a special start up state with the tape head on tape cell number one. The input resides on the first few tape cells (as many as are needed) and the remainder of the tape is initially blank.

The TM might seem at first glance to be too simple to represent all feasible computations and it may not seem to look much like a modern computer with its internal memory, I/O ports and so on. So let us present a second model (see [1]) known as a RASP, or Random Access Stored Program model of computation. In this model we are allowed an (infinite) array of memory cells, the program is stored in the memory along with its input, and the program is run by a fixed CPU with a simple instruction set and a program counter. In other words, the RASP looks a lot like a very simplified desk top computer.

Now, the key insight is that each of these models can be represented as a simple mathematical system. These in turn can each be represented as a fixed program on either of the machines which take as input the description of a machine program and its input and will emulate, or run these programs in exact faithfulness to its intended machine. This is no different in principle than running an emulator for an IBM PC on a MacIntosh in order for example to run Windows 95.

Finally, it is an easy exercise to write a Turing Machine emulator on your favorite personal computer despite claims [24] it is not "physically realizable". Now of course you will not be able to provide an infinite tape but you can do the following. Let the program store the tape representation on diskettes. For a few billion dollars you will be able to provide lots of memory to the machine. If you have ever attempted one of those installations with the "insert disk n" sequence of instructions, you will realize that it is possible to emulate as close to infinity as you will have the patience to endure. This also illustrates that the requirement of having the remainder of the infinite tape initially blank is a mere mathematical convenience, since a realizable alternative is to simply initialize memory as it is required.

We have departed somewhat from our objective of discussing the *complexity* or efficiency of algorithms, since the topic of TMs and the Halting Problem are really about what is computable. For complexity issues we are no longer concerned about infinite resources, only the way in which resource requirements increase as a function of input or problem size. In terms of memory this is exactly our "diskette" model where the available hardware resources are bounded by the available money. Since we no longer require the TM to be infinite (except for mathematical convenience) the TM is realizable for bounded (but arbitrary) sized problems.

The second resource we are interested in is execution time, which in our abstract model is measured by the number of state transitions made by the TM before it finishes.

It turns out that in fact a TM emulation will only increase the running time of a RASP by a polynomial amount; in fact if the RASP uses $T(n)$ time then the TM emulation will use approximately $O(T^3(n))$ time. (See for example figure 1.6 of [15]. A RAM is very much like a RASP, the difference is discussed in [1].)

Thus, if a particular algorithm requires exponential time on a TM then it will also require exponential time on a RASP, and if it can be implemented in polynomial time on a RASP then it can be implemented in polynomial time on a TM. The reader may wish to substitute "my favorite computer" for RASP in the preceding statement.

Finally, we want to be clear that we are most concerned with classes of *problems*, not classes of algorithms. For example, we might describe an "optimization class" by requiring as input a description of a function, and require as output an instance of the domain that maximizes the function. This class is related to the very broad class addressed by the NFL. We will return to this later, but first let us introduce two of the most famous problem classes, P and NP.

For reasons of mathematical simplification these classes are both only defined over *decision* problems; that is, problems whose answer is "yes" or "no". We will relate these to more general optimization versions of the problems below when we discuss NP-easy and NP-hard classes.

The class P consists of all those decision problems for which there exists an algorithm that can answer the query in polynomial time.

The term NP is an unfortunate acronym, due to historical development, that is often confused by the uninitiated and incorrectly interpreted to mean "non-polynomial". In fact, quite to the contrary NP refers to a class of decision problems that have polynomial time solutions but on something called *a non-deterministic Turing machine*(NDTM). Thus, NP stands for "non-deterministic polynomial". It is also important to note that "non-deterministic" *does not mean* "random" or "probabilistic". Furthermore, we cannot implement a polynomial time non-deterministic TM, and in fact that restriction is fundamental to the distinction between P and NP. If we could implement this strange TM then we would have P=NP and our conversation would stop. The purpose of the NDTM is solely to identify an interesting class of problems.

Okay, a little confusion on the part of the reader at this point is understandable. Instead of NP the reader may wish to substitute the acronym PV which we will define to mean "polynomially verifiable." PV is the class of decision problems for which all the instances having "yes" answers also have example solutions with simple proofs that can be verified in polynomial time.

The idea is that the classes P and PV basically capture the distinction between the difficulty of solving a problem from scratch versus the difficulty of checking that someone else's solution is correct. A moments thought should convince you that P must be a subset of PV since if we can solve the problem easily from scratch, then to verify someone's answer we can ignore their proof and check our solution against theirs.

It turns out (see [4] for example) that PV $\equiv$ NP. It is undoubtedly too late to change the name throughout the world but the reader is encouraged to make the substitution of PV for NP while reading this document.

As an example, consider the infamous Traveling Salesperson Problem (TSP). A problem instance consists of a set of cities, the distances between the cities and a tour cost bound B. The yes/no question is "does there exist a cyclic tour visiting each city exactly once with total cost at most B?" To show that this belongs to NP, we need only note that if we are given a particular tour then it is easy to run through it and compute the total cost and then compare whether this cost meets the bound B. It is this algorithm for verification that defines which problems are members of NP.

The unproven assumption underlying almost all of complexity theory is that the inclusion P $\subseteq$ NP is proper; that is, that there are problems which can be easily verified given a solution but which cannot be easily solved in the absence of a solution. In this context "easy" means there is a polynomial time algorithm while "hard" means that the best possible algorithm will be exponential.

It also turns out that TSP is a member of a particular subclass of NP, called NP-complete. All the problems in this class have the property that an algorithm which could solve any one of them in polynomial time could be used as a subroutine to solve any other problem in NP in polynomial time. In this sense these are the hardest problems in NP.

It is beyond the scope of this paper to even outline the proof but one crucial feature needs to be mentioned. That is that the proof depends on the verification algorithm. In order to use the (mythical) polynomial time TSP algorithm (or algorithm for any other NP-complete problem) as a subroutine to solve an arbitrary problem in NP, the arbitrary problem must the transformed into a different form. There is a standard method[1] for doing this but this method requires the verification algorithm. We will see the importance of this when we relate NFL to NP in the next section.

The conjecture, supported by many years of research that so far has failed to find any counter example, is that P is not equal to NP. Until shown otherwise, it should be considered as a "law" of the computational universe that no NP-complete problem has a polynomial time solution and any claim to having a program be it GA or other, that efficiently "solves" an NP-complete problem should be very carefully qualified. A claim that purports to efficiently solve an NP-complete problem means the author is claiming that P=NP and this requires a very formal proof. Otherwise, the claim should be considered algorithmic nonsense.[2]

In particular, note that finding an optimal solution to a particular instance of an NP-complete problem, or even on average over some distribution of instances does not solve the problem in general. It is well known that many NP-complete problems have large classes of instances that can be solved easily by simple algorithms. On the other hand, there are empirically identifiable subclasses that are extremely difficult even for randomized algorithms, or for approximation algorithms. Before proclaiming the efficiency of an algorithm, experimenters are urged to check their algorithms on instances of these classes. In most cases it will prove to be a humbling experience.

Now let us consider the corresponding TSP optimization problem: given a set of cities and distances, what is the optimum cost tour? Notice that if we could solve this problem then we could easily solve the correspond-

[1]The 'method' is reduction to SAT, see [15]

[2]There is some evidence that quantum computers may be able to polynomially solve some problems not in P, but not NP-complete problems [4]. Practical quantum computers have yet to be implemented.

ing decision problem and so this problem must be as hard as the decision problem. We say that the optimization version of an NP-complete problem is NP-hard. Everything we have said about the difficulty of solving NP-complete problems applies also to NP-hard problems.

On the other hand suppose we had an algorithm to solve the TSP decision problem efficiently. Now we can use the decision solver to find an optimal TSP solution as follows. First we will generate any tour at random and compute its cost T. We will then do a binary search to find the minimal B in the range $1 \ldots T$ for which there is a tour of cost B. At each step we will ask the decision problem whether there is a tour of cost B. This takes polynomial time. (Total cost $O(p(n) \log_2 T)$, where $p(n)$ is the cost of solving the decision problem.) Having found the minimal B, we can find an optimal tour as follows. For each pair of cities in turn change the intercity cost to $B+1$. Now ask whether there is a tour of cost B. If so then the edge is not required in the optimal tour. Otherwise it is, and the cost has to be reset to the original cost before the next step. In this way, we can solve the TSP optimization problem if the decision problem is solvable in polynomial time. For such cases we say the optimization problem is NP-easy.

It turns out that almost all of the interesting NP-hard optimization problems are also NP-easy. In other words, these optimization problems are, up to a polynomial factor, equivalently as hard as solving any problem in NP.

Finally, one may get the impression from [24, 20] that because GAs or other optimization techniques are randomized and may only be required to give approximate solutions, they somehow escape the complexity theory that has been developed. There are two responses to this notion.

First, there are numerous classes of problems that do include randomization as part of their description. For example, informally the class RP allows for the input of a random stream of bits which can be used in the algorithm in the same way that any pseudo-random number sequence is used in a GA or other adaptive algorithm. This class includes all (decision) problems that can be solved with probability better than 1/2 in polynomial time. (The actual definition is a bit more technical - See [4]). This class definitely includes all of P and is included in NP but it appears that NP-complete problems are not included in RP. Until shown otherwise we may assume that randomization cannot solve NP-complete problems.

Similarly, there are numerous results on approximations to NP-hard optimization problems. Recently for example it has been shown that unless P=NP it is not possible to have a polynomial time algorithm that will approximate the graph coloring problem (next section) to within any constant factor. (In fact, for $\epsilon < 1$ it cannot be approximated to within $O(n^\epsilon)$.)

One final class we mention in the next section needs a brief mention. This is the class PSPACE with its attendant PSPACE-complete subset. This is the class of problems which can be solved using only polynomial memory. This includes NP as a subclass and again the inclusion is thought to be proper. To get some intuition of what can be done in polynomial space consider that the number of distinct states of a desk top computer with 32 megabytes of memory is roughly $2^{256000000}$. If a computer were to step through all possible states at its maximum possible speed it would take many times the current estimate of the age of the universe to finish.

6 Relating NFL to Standard Complexity Theory

We would like to relate blind search models to standard complexity classes such as P, NP and PSPACE [15], and so to relate NFL-like consequences. The NFL theorems allow a lot of freedom to the adversary generating the functions. In particular, the adversary must have exponential space to store the function values. (This is a direct consequence of Kolmogorov complexity [19] – the "average" random or arbitrary function is not compressible. That is, the shortest possible universal description is to list the function values in order of some enumeration of the domain.) The adversary can evaluate the strings in polynomial time however, since fast access of a string can be done by storing the strings in a digital tree.

Suppose we restrict the adversary to polynomial space and time (the space restriction seems to be more important). This enormously reduces the set of functions that can be used against our algorithms. Clearly, most "interesting" functions will still be in such a class, as well as many uninteresting functions.

Now, the question we would like an answer to is "Does the NFL theorem apply to this restricted class?". That is, does it still require exponential time on average to optimize a function regardless of the algorithm applied.

Before answering this, let us consider some of the possible consequences if the answer turns out to be yes. (Actually, to thwart those who jump ahead to look for the answers, I don't have an answer!)

We will consider a simple example problem that is known to be NP-hard and thus strongly suspected of not being tractable. The problem we will consider is the graph coloring problem. Given a graph $G = (V, E)$, we are asked to provide an optimal coloring of it. A c-coloring of a graph is an assignment of integers from the range $[1..c]$ to the vertices of the graph so that no two vertices which are joined by an edge have the same color. The coloring problem is to find a coloring that minimizes c.

Being NP-hard means that we do not expect any algorithm to be able to solve arbitrary instances of the graph coloring problem to optimality in polynomial time. Of course, many classes of graphs can be colored in polynomial time, or at least expected polynomial time, but there remain subclasses that appear hard to all known algorithms [12, 10].

Now suppose we wish to set this problem in the context of our blind search model. Note that for the usual definition of this problem we are given as input the graph G and asked to find an optimal coloring of the graph.

To cast this problem to the blind search model we need to describe a function that is evaluated by the black box. One way of doing this, one that also seems to be favored by many in the GA community[13], is to choose as our function domain permutations of the set of vertices, and then have the environment evaluate the permutation by coloring the permutation using the simple greedy graph coloring algorithm.

The greedy algorithm takes the vertices in the given order and assigns to each vertex in turn the minimal color that does not conflict with vertices joined to it by an edge and already assigned a color. The reader is referred to [11] for a more complete description of this algorithm and its performance.

The adversary must return an integer representing the number of colors used by the greedy algorithm. In addition we may choose to require our adversary to return the colors assigned to each vertex. Returning more information is another way that an algorithm may be made stronger than is allowed under the blind search model.

Note that if the graph has k vertices, then the length of a binary string representing a permutation is $n \approx k \log k$ bits.

Note the following possible classifications with respect to "blindness" or lack of knowledge:

1. Full knowledge: The algorithm knows G and thus is free to choose its permutations taking into account the complete structure of the graph.

2. Partial Knowledge: The algorithm knows it is faced with a graph coloring problem on k nodes but the graph is hidden from the algorithm. *In this model, the adversary is free to make up the graph as it goes, provided its colorings are consistent.*

3. Little Knowledge: The algorithm knows that it is faced with a problem from NP, and perhaps even that the $n \approx k \log k$ bits represent a permutation of some domain, but it does not know which problem it has.

4. Almost no Knowledge (Almost blind search): The algorithm knows only that it is faced with minimizing a function from strings of length $n \approx k \log k$ bits to some integer domain. We will grant that the adversary is using only polynomial space and time to represent the function.

5. No Knowledge (Blind search): The algorithm knows only that it is faced with a function from strings of length $n \approx k \log k$ bits to some integer domain that must be minimized. This is just the blind search class that the NFL theorems prove we cannot search with polynomial efficiency.

In the first instance, the algorithm might for example use a sophisticated backtrack algorithm to compute an optimal coloring, then send a permutation to the black box that generates that coloring under greedy. Thus, under the black box model the algorithm runs in *constant time* since only one string is evaluated.

The theory and conjectures of NP-completeness however, say that it is unlikely that the actual computation required to find this string can be carried out in less than exponential (in k) time. This is one of the senses in which NFL-like theorems are much weaker than standard complexity theory, although the NFL results are *theorems* while we confess that P$\neq$NP remains a conjecture.

Intuitively, the second knowledge class restricts the power afforded an algorithm, or equivalently extends more power to the adversary. Any algorithm will only be able to choose permutations to send to the black box based on previous evaluations. These evaluations may include color assignments to the vertices as well as the number of colors used. Any information about the structure of the graph will have to be inferred from the coloring information returned.

However, it is likely closer to the model that many have in mind when applying EAs to a problem. Some intelligent use can be made of the colors returned since vertices of the same color must form independent sets of vertices. An independent set is a set of vertices with no edge between any pair. Thus, for example, recombination operators might try to preserve and remix those independent sets. Such considerations led to the iterated greedy algorithm described in [11]. (Originally, the iterated greedy algorithm grew out of an attempt to apply a GA to coloring.)

The third knowledge class yields far more power to the adversary and provides us an interesting comparison of NFL to the classes P, NP and RP. If we could solve every problem in the third class in polynomial time with a deterministic algorithm then we would have shown P = NP. If we could solve any problem using a randomized algorithm then we would have NP = RP (or reduce NP into BPP depending on the particulars. See e.g. [4] for more information). Neither of these is considered likely.

However, there is a more subtle distinction to be made here. We know that if any NP-complete problem could be solved in polynomial time then P = NP or RP = NP depending on the nature of the solution as above. Does this mean that the third knowledge class is equivalent in difficulty to the NP-complete class?

Certainly, this class must be as hard as the NP-complete class. But, solving an NP-complete problem either probabilistically or deterministically would not appear to solve the third knowledge class. The reason is that for a problem X to be in NP, X must come with a verification algorithm. That is, in general there must be some method of specifying a solution together with an algorithm that proves the solution whenever the instance has a solution. For graph coloring for example, to prove that a graph is k-colorable we need only specify a coloring of the graph, then it is easy to write a polynomial time program to verify whether such a coloring is legal and satisfies the constraints.

Given an arbitrary problem X in NP and a solver for some NP-complete problem Y, the problem X can be solved by constructing an algorithm that uses the Y-solver as a subroutine *but the construction depends critically upon having the verification program for X.* On the other hand, in the third knowledge class listed above, we do not know the exact problem so we cannot have this certification program available.

The bottom line is that solving all problems in NP at once with a mostly blind search is likely much harder than solving any specific NP-complete problem. That is, mostly blind search is not only a much weaker approach to solving a specific problem than is one using problem specific knowledge, but also a weaker approach to solving NP in general.

In the fourth knowledge class, not only do we not know the problem at hand, we have made an even weaker assumption as to the usual complexity classes. The adversary is restricted to polynomial space in which to record the function values. The space restriction is to disallow the adversary simply building a digital tree representation of the discrete domain, which would allow an essentially arbitrary function to be constructed by randomly choosing domain values on demand. Without this restriction, the adversary would be very close in power to the adversary we used in our version of the NFL theorems. This class not only includes all of NP, but apparently all of PSPACE, and the associated optimization functions.

In summary, conjectures on the nature of NP and PSPACE suggest that these problem classes are intractable. It is difficult to see how throwing away information and enlarging the problem classes with which an algorithm must cope can improve the search capabilities. The NFL theorems are much weaker than usual complexity in the sense that the claims of difficulty are made on much larger classes of problems. Or alternatively, the restrictions on overall EA effectiveness implied by complexity theory are much more severe than those implied by NFL.

Part III
So What?

7 On Genetics as a Universal Search Engine

[3] If you take a LISP program and run it through your favorite C language compiler, you will probably not get much in the way of useful output. If you take a binary executable from an old 6502 based Apple computer and try to run it on your 486 based PC, the machine will probably not respond.

Similarly, if you take a spoonful of raw DNA and dump it on the sidewalk, the results will likely be less than spectacular. The universe as a general rule simply does not know how to interpret DNA, anymore than it knows how to interpret a punched paper tape without a teletype.[4] A DNA code only has meaning within the context of an appropriate device, namely the cell for which it is fitted. (Viruses, which get interpreted within someone else's device, can be thought of as being fitted to those devices.) This is one of the themes explored by Cohen and Stewart [7].

[3] Biologists and others familiar with real genetic evolution may wish to take two aspirin before reading this section. My apologies in advance for my superficial view of this science.

[4] Unlike the paper tape, the DNA may replicate when dumped on the sidewalk, if temperature and other conditions are appropriate. But it is unlikely to produce a dinosaur in the absence of an egg cell.

Clearly, for evolution to have happened the interpretive device and the code had to co-evolve. It seems entirely reasonable that quite different devices might have evolved, and that quite different DNA sequences would then have been required to be properly interpreted within them. Thus, it seems intuitive that perhaps the DNA code we have is really quite arbitrary.

And now the rub. For if DNA coding is arbitrary, and if as Cohen and Stewart [7] suggest, genetic evolution is a kind of computation over the universe, then how, in the face of NFL, did it it ever get so far?

Note that we are talking about strings billions of characters in length. And we are talking about a system where a single mutation may cause the offspring, if it develops at all, to be completely ineffective. It would appear that mutation is effectively doing a blind search when it arbitrarily modifies DNA.

In fact it seems that nature believes that mutation is doing blind search as well. According to a popular source [17], during human reproduction only about one in a billion genes will undergo mutation. It seems that very complex test and repair mechanisms within the DNA reproductive environment keep the mutation rate exceedingly small.

So if mutation alone were responsible, generating a billion character string one character at a time would take *at least* a billion generations, which is considerably more than evolutionary history permits for humans. "But", the response comes back, "it is the recombination operators that allow small pieces to be put together and form the really long sequences." So we invoke the NFL to argue that if the operators are blind there is no reason to believe that crossover could do any better than mutation.

Yet, in contrast to mutation, nature not only allows crossover to operate but seems to glory in it. Most higher species are divided into two distinct sexes and an elaborate system has been established with the apparent sole purpose of ensuring that every generation initiates the mixing of genetic material for the next generation.

The only reasonable conclusion seems to be that nature does not see crossover as a blind search operator.

8 Partial Information and Myopic Search

Let us revisit our two friends on a park bench trying to solve Rubik's cube in the introduction. Now let us suppose that instead of being blind, the searcher is only perhaps color blind, or of very poor eyesight. In particular, notice that the searcher will know the general structure of the cube and its basic operations. Also, let us suppose that the friend is a little less miserly in the information he is willing to impart. Suppose the response to the queries takes on the form "you have 5 red cells on the red face, 3 blue cells on the blue face,.." and so on. In this way the searcher has more information to guide his search. He may not know which face is the blue one, but as he performs more probes, he will gain more insight and so be able to give his search some direction.

For a simpler example, in the game of Mastermind [5], one player, call her the adversary, hides some colored pegs under a little cover. The other player, the searcher, makes probes by filling in a corresponding set of holes with pegs. After each probe, the adversary tells the searcher how many pegs match the color of the hidden pegs and how many matches are in the same position. However, the adversary does not state which pegs or colors are matched. Again, the searcher must use the partial information to generate future probes that will provide maximal information. Notice that in a simple variation the adversary could make up her answers in a consistent manner after each probe, thus potentially increasing the number of probes required to narrow the search until only one possible matching pattern remains.

Other games with partial information abound, and some are also stochastically based, such as poker. Is there a place here for the use of adaptive algorithms in search that is not quite blind but rather myopic?

Could it be that the reason that nature uses crossover is because it has already learned that the things coded for in the segments of DNA over which it allows crossover are useful? At first glance this appears to be a rather strong form of the building block hypothesis [16] but the reader is cautioned that DNA interpretation is not merely a matter of decoding linearly separable bits of DNA. Rather there appear to be structural hierarchies within DNA. For example, some segments can do things such as inhibit or activate other segments.

One thing we should keep in mind is that the universe does not appear "random" in the sense that all possibilities are equally likely. It does satisfy quite strong natural laws, and in particular the earth seems to stay within quite narrow bounds on variables such as climate, chemical composition and so on.

So, assuming the absence of divine intervention, somehow nature started with a nearly blind search. It slowly learned some chemical tricks, slowly discovered they could be encoded into DNA, and then slowly began building devices for recombining the DNA sequences to provide ever better search strategies based on what it had learned so far. Mutational error rates went down, as those systems allowing errors seldom survived. On the other hand, recombination, because it was becoming ever better tuned with the representation, was enhanced. Basically, crossover, inversion and other genetic operations tended to mix sequences that already coded for something useful within the device. Intuitively, mutation would tend not to.

If this picture is even remotely correct then part of the fitness of the genes that DNA encodes must be their ability to continue to carry useful information when mixed by the recombination operators. That is, not only must the encoding device evolve with the DNA but more particularly so must the operators. Once the search was no longer blind, that is as knowledge was built into the search system, then evolution could be enormously speeded up.

Now, if the operators are no longer blind but merely myopic, then they can no longer have universal application. In a strong sense, since a searchscape is in part defined by the operators and representation, the landscape being searched by evolution must also be evolving along with the operators, devices and DNA. But how can life continue to increase in fitness if the evolution landscape has been restricted to some subset based on known tricks? That is, if it has been to some extent decoupled from the background world it is evolving in? This brings us to the next big question, which is "*what*, if anything, *is* this *fitness* that evolution is supposed to increase?"

9 The Competitive Edge

Genetic search is not necessarily optimizing. It may only be competing in local events.

For example, consider the following thought experiment. It would seem that the best way to collect sunlight while minimizing overhead costs in materials used would be to have a large flat leaf hugging the landscape. Or perhaps, if not a single leaf, then something akin to a low cover of moss. Trees are not close to optimal in this sense because they must invest in a tremendous infrastructure and much of it will be shaded and so not productive.

Nevertheless, nature seems to favor trees. The reason is not that they can gather more sunlight (only so much hits the earth and the flat leaf could get it all) but rather that they prevent flat leaves from getting any sunlight. Thus, they have a competitive edge over flat leaves. In order to survive you only need to compete with those next to you. This is not necessarily consistent with any global idea of optimization or fitness.

So, is it possible then that evolution is not optimizing at all? Could it just be wandering around in some sort of empty-headed mutual bashing contest?

And finally, is the application of the NFL, indeed of the computational model to natural evolution meaningful, or mere sophistry?

10 Bad News/Good News

So what does all of this say about using genetic evolution as a model of search in optimization settings? Basically it says that the faith in using a GA as a blind search optimization engine is misplaced. It is based on an understanding of genetic evolution that does not necessarily apply within a computational optimization setting.

In other words, there is no reason to believe that a genetic algorithm will be of any more general use than any other approach to optimization. And in particular, if a GA is going to be used, then the user will have to perform an analysis of the problem within the context of the GA and determine operations and representations that are compatible with the problem and will enhance the search. If heuristic methods are sufficient for the researchers problem then again there is no a priori reason to choose a GA over other standard heuristic programming methods that might lend themselves to the problem at hand.

In fact, due to the complexity of interactions within a GA, this approach may well be more difficult on average than simply trying to solve the problem using conventional methods. With an evolutionary algorithm, the algorithm itself represents something of a black box with complex internal interactions between various operators and multitudes of parameter adjustments to the system with no clear and detailed understanding of how they interact. The researcher trying to solve a problem is then placed in the unfortunate position of having to find a representation, operators and parameter settings to make a poorly understood system solve a poorly understood problem. In many cases he might be better served concentrating on the problem itself.

Occasionally there may be problems where the GA paradigm is superior. In particular, for some problems the notion of combining results from various local searches to direct future search seems appealing. And hacking around in a semi-formal, semi-intuitive fashion sometimes yields reasonable first results, along with new insights. Furthermore, even when other methods are carefully tuned to certain problems or instances, when these methods are combined the efficacy of the results are often hard to predict. For example, combining different heuristic local search graph coloring algorithms sometimes produced unexpected negative performance [10]. Thus, the complexity of GA internal interactions is not in itself sufficient reason to abandon further research.

Much further basic research will be required to determine where GAs are most applicable. The fact of natural evolution does not indicate where these areas of applicability might be, and it certainly yields no basis to claim GAs as a universal magic bullet. This message may be bad news or good news depending on whether you are looking for a quick fix problem solver, or an interesting and difficult domain for further research.

References

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Company, Inc., 1974.

[2] Lee Altenberg. The schema theorem and price's theorem. In L. Darrell Whitley and Michael D. Vose, editors, *Foundations of Genetic Algorithms 3 (FOGA 3)*, pages 23–49. Morgan-Kaufmann, 1995.

[3] David L. Battle and Michael D. Vose. Isomorphisms of genetic algorithms. In Gregory J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 242–251. Morgan Kaufmann, 1991.

[4] Gilles Brassard and Paul Brately. *Fundamentals of Algorithmics.* Prentice-Hall, Inc., 1996.

[5] V. Chvátal. Mastermind. *Combinatorica*, 3:325–329, 1983.

[6] Daniel I. A. Cohen. *Introduction to Computer Theory.* John Wiley & Sons, Inc., New York, New York, revised edition, 1990.

[7] Jack Cohen and Ian Stewart. *The collapse of chaos : discovering simplicity in a complex world.* Viking, New York, 1994.

[8] L. Comtet. *Advanced Combinatorics.* Reidel, Dordrecht, Holland, 1974.

[9] Joseph Culberson. Mutation-crossover isomorphisms and the construction of discriminating functions. *Evolutionary Computation*, 2(3):279–311, 1994.

[10] Joseph Culberson, Adam Beacham, and Denis Papp. Hiding our colors. In *CP'95 Workshop on Studying and Solving Really Hard Problems*, pages 31–42, Cassis, France, September 1995.

[11] Joseph C. Culberson. Iterated greedy graph coloring and the difficulty landscape. Technical Report TR 92-07, University of Alberta Department of Computing Science, Edmonton, Alberta, Canada T6G 2H1, 1992. ftp ftp.cs.ualberta.ca pub/TechReports.

[12] Joseph C. Culberson and Feng Luo. Exploring the k–colorable landscape with iterated greedy. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1995. to appear: preprint available http://web.cs.ualberta.ca/~joe/.

[13] Lawerence Davis, editor. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 1991.

[14] K.A. DeJong. *Analysis of Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, The University of Michigan, 1975.

[15] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[16] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Inc., 1989.

[17] Larry Gonick and Mark Wheelis. *Cartoon guide to genetics.* Barnes and Noble, 1986.

[18] John H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[19] Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and its Applications.* Springer-Verlag, 1993.

[20] William G. Macready and David H. Wolpert. What makes an optimization problem hard? Technical Report SFI-TR-95-05-046, The Santa Fe Institute, Santa Fe, New Mexico, February 1996. ftp://ftp.santafe.edu/pub/wgm/.

[21] William M. Spears (Moderator). Yin-yang: No-free-lunch theorems for search. Panel Discussion at International Conference on Genetic Algorithms (ICGA-95), July 1995. http://www.aic.nrl.navy.mil:80/~spears/yin-yang.html.

[22] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, Inc., 1994.

[23] A. M. Turing. On computable numbers with an application to the entscheidungs-problem. *Proceedings of the London Mathematical Society 2*, 42:230–265, 1936. *ibid.* 43, 544-546.

[24] David H. Wolpert and William G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, The Santa Fe Institute, Santa Fe, New Mexico, February 1996. ftp://ftp.santafe.edu/pub/wgm/.