Proceedings of the 40th IEEE
Conference on Decision and Control
Orlando, Florida USA, December 2001

Simple Explanation of the No Free Lunch Theorem of Optimization

Yu-Chi Ho and **David L. Pepyne**

Division of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138
ho,pepyne@hrl.harvard.edu

Abstract

The No Free Lunch Theorem of Optimization (NFLT) is an impossibility theorem telling us that a general-purpose universal optimization strategy is impossible, and the only way one strategy can outperform another is if it is specialized to the structure of the specific problem under consideration. Since virtually all decision and control problems can be cast as optimization problems, an appreciation of the NFLT and its consequences is essential for controls engineers. In this paper we present a framework for conceptualizing optimization problems that leads useful insights and a simple explanation of the NFLT.

1. Introduction

Many scientific fields of study have postulated impossibility theorems. In mathematics, for example, Godel's theorem roughly states that in any mathematical system facts always exist that cannot be proved or disproved. In economics, Arrow's Impossibility Theorem on social choice precludes the ideal of a perfect democracy. The No Free Lunch Theorem (NFLT) [1,2,5,7,8,10-14], though far less celebrated and much more recent, tells us that if we cannot make any prior assumptions about the optimization problem we are trying to solve, no strategy can be expected to perform better than any other. Put another way, a general-purpose universal optimization strategy is theoretically impossible, and the only way one strategy can outperform another if it is specialized to the specific problem under consideration.

Without question, optimization is central to the decision and control sciences making an appreciation of the NFLT and its consequences fundamental. Our contribution in this paper is to provide what we believe is a simple and intuitive explanation of the NFLT and its implications. Our main assumption is one we call the *finite world assumption*, where all input and output sets are assumed to be discrete and finite in size. The finite world is the world of digital computers, and hence does not really impose any loss of generality since virtually all optimization nowadays is done using digital computers. In a finite world all the information about an optimization problem can be summarized in a matrix we call the **P**-matrix (for problem matrix). In its most broad interpretation, the rows of the **P**-matrix are strategies, the columns the universe of all possible problems, and the entries performances of the strategies on the problems. The essence of the NFLT is that the row averages in a **P**-matrix are always equal, i.e., averaged over all possible problems, all strategies give the same performance. Deeper structure of the **P**-matrix extends the result to search algorithms and leads to many other useful insights. We point out, that although our presentation here is informal, all of results can be made rigorous.

Funding for this work was provided by EPRI/DoD CIN/SI contract WO8333-03, Army contracts DAAL 03-92-G-0115 and DAAH 04-0148, AFOSR contract F49620-98-1-0387, ONR contract N00014-98-10720, and a DoD CIP/IA Fellowship.

2. Problem Formulation

In optimization we are concerned with mappings of the form $y=f(x)$, where x is a candidate from a "solution" set X, and y is a scalar from a "performance" set Y. The optimization objective is to choose the solution $x \in X$ whose performance y is "best" in some sense (e.g., minimizes f). In a *finite world* the sets X and Y are discrete and finite in size representing, for example, the input and performance spaces of a discretized continuous-valued optimization problem or the set of tours and tour lengths of the combinatorial Traveling Salesman Problem (TSP). When the sets X and Y are finite, then the set of possible mappings from X to Y is also finite. In particular, if X has size $|X|$ and Y has size $|Y|$, then by direct enumeration there are $|F|=|Y|^{|X|}$ unique mappings in the set $F=\{f: X \rightarrow Y\}$.

2.1 *The P-matrix*

Since the sets X, Y, and F are all discrete we can assign an integer label to each of their elements, i.e., let $X=\{x_0,x_1,\ldots,x_{|X|-1}\}$, $Y=\{y_0,y_1,\ldots,y_{|Y|-1}\}$, and $F=\{f_0,f_1,\ldots,f_{|F|-1}\}$. How we label the elements is entirely arbitrary, as we will see later. We now define the **P**-matrix as the matrix with $|X|$ rows labeled with the elements of X, $|F|$ columns labeled with the elements of F, and i,j-th entry equal to $f_j(x_i) \in Y$. As

an example, if $|X|=3$ and $|Y|=2$, then $|F|=|Y|^{|X|}=8$ and the **P**-matrix is given by,

	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7
x_0	y_0	y_1	y_0	y_1	y_0	y_1	y_0	y_1
x_1	y_0	y_0	y_1	y_1	y_0	y_0	y_1	y_1
x_2	y_0	y_0	y_0	y_0	y_1	y_1	y_1	y_1

(1)

In practice, the size of the input set $|X|$ is often exponential in the problem size (i.e., the dimension of a discretized real-variable problem or the number of cities in a TSP). Likewise, $|Y|$ is also often huge. This generally precludes actually constructing the **P**-matrix, storing it in memory, and using it to solve optimization problems. As a conceptual framework, though, the **P**-matrix can provide useful insights and simplify proofs.

2.2 *A Fundamental Counting Lemma*

The **P**-matrix is a generalization of a class of matrices we call *counting matrices*.

Definition 2.1: For a given pair of positive integers O and I, the I by O^I matrix whose columns are obtained by counting in base O from 0 to O^I-1 is called a **counting matrix** C.

For example, the counting matrix for $I=3$ and $O=2$ is obtained by counting in base-2 (binary) from 0 to 7, i.e.,

$$C = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (2)$$

Counting matrices have the following key properties,

LEMMA 2.1 (Counting Lemma): *For a counting matrix (defined by a given pair of positive integers O and I):*

a) *Each integer* $0,1,\ldots,O$ *appears* O^{I-1} *times in each row.*
b) *Pick any row i and any entry* C_{ij}. *The submatrix formed by eliminating row i and all columns k such that* $C_{ik}\neq C_{ij}$ *is a counting matrix with I-1 rows and* O^{I-1} *columns.*

Proof: The proofs are not hard, so lacking space we will only be informal here. Regarding a), it is clear its hold for the counting matrix in (2), and a few examples should convince you that it holds for counting matrices of any size. To illustrate b), begin with the counting matrix in (2) and form a submatrix by eliminating row 1 and all columns where the row 1 entry is not 1. Doing this gives,

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad (3)$$

which is the counting matrix associated with $I=2$, $O=2$. Note that this property is recursive. ♦

Property a) leads to an immediate corollary.

Corollary 2.1: *For a counting matrix, all row sums, and hence row averages, are equal.*

Now note that the matrix whose entries are the *integer labels* of the y's in the **P**-matrix in (1) is precisely the counting matrix in (2). In other words, ***P-matrices are structurally equivalent to counting matrices.*** Hence, **P**-matrices satisfy the counting lemma (with $O=|Y|$ and $I=|X|$) regardless of the values associated with the y's.

3. The No Free Lunch Theorems

The **P**-matrix and counting lemma are at the heart of our explanation of the NFLTs.

3.1 *Optimal Strategy Selection*

In the most general setting, the x's (rows of the **P**-matrix) are *strategies* and the f's (columns of **P**) are all possible optimization *problems*. In this most general setting, a strategy is a mapping from the space of available information to a control variable or decision space. Strategies include methods involving search, adaptation, learning, voting, feedback, dynamic programming, evolution, randomization, and even humans in the loop. In short, the concept of strategy covers any method for coming up with a solution to an optimization problem. Nothing can be more general or more inclusive. That all row averages in a **P**-matrix are equal leads immediately to the NFLT, *averaged over all problems, all strategies have the same performance.* Or put another way, *it is impossible to develop an optimization strategy that is universally better than all others.*

3.2 *Function Optimization*

A more specific and familiar setting involves function optimization problems of the form,

$$\min_{x\in X} f(x) \quad (4)$$

where the x's are vectors and the f's are different objective functions. The usual strategy for solving (4) is via search. The NFLT for search algorithms that halt after picking m distinct samples from X is given in [8,13,14]. One interpretation of the NFLT for search is that *unless you can make prior assumptions about the* $f\in F$ *you are working on, then no search strategy, no matter how sophisticated, can be expected to perform better than any other.* Proving this result is complicated by the fact that it must be established for *all* algorithms, including those that learn and adapt themselves based on the performances they observe as they sample. Using the **P**-matrix, however, the result can be made intuitive. In particular, suppose you have the entire **P**-matrix available to use in guiding your search for the

optimal solution. Then the most powerful strategy you could develop would be one that works as follows. Each time you take a simple x_i and observe performance y, eliminate from the **P**-matrix row i and all columns j for which $f_j(x_i)\neq y$. Intuitively, one should think that such a procedure should allow you to quickly identify which $f\in F$ you are working on, at which point you can pick the optimal row by inspection of the **P**-matrix. However, because each of the submatrices formed by the above procedure are themselves (smaller) **P**-matrices, each of the remaining rows all have equal average performance over the remaining columns. Hence, there is nothing you can learn from this procedure that will help you identify the optimal any more efficiently than any other procedure. In other words, when you know nothing and must assume that all $f\in F$ are equal likely, then all search strategies are on equal footing.

The NFLT for search algorithms leads to some rather surprising and counterintuitive results (see also [8,13,14]). Since on average all search algorithms give the same performance, the surprising result is that *on average all search algorithms perform no better than random search.* This means that unless we can make some prior assumptions about the $f\in F$ we are working on, no search algorithm we choose can a priori be expected to perform any better than random search, and the risk we take is that it might actually perform *worse* than random search! A counterintuitive result is that when averaged over the universe of all objective functions *hill-climbing* actually performs the same as *hill-descending* even when the goal is function minimization!

3.3 *Stochastic Optimization*

Sometimes we are not able to evaluate the objective function in (4) exactly, but rather we have $y=l(x,\omega)$, where $x\in X$ is a candidate solution and $\omega\in\Omega$ is a random quantity reflecting uncertainty or error in our ability to evaluate its performance $y\in Y$. This leads to the stochastic optimization problem,

$$\min_{x\in X} E_\Omega[l(x,\omega)] \tag{5}$$

where E is the expectation operator. Again, in a finite world the spaces X, Y, and Ω are discrete and finite. Thus, if the probability mass function defined over Ω is stationary, then conceptually a stochastic optimization problem is identical to the deterministic optimization problem,

$$\begin{aligned}\min_{x\in X} f(x) &= \min_{x\in X} E_\Omega\left[l(x,\omega)\right] \\ &= \min_{x\in X}\left[\sum_{\omega\in\Omega} l(x,\omega)p(\omega)\right]\end{aligned} \tag{6}$$

where $p(\omega)$ is the probability mass associated with ω.[2] This observation immediately extends the NFLT in Section 3.2 to stochastic problems.

3.4 *Input Representations and Neighborhood Structures*

The performance of certain algorithms (e.g., genetic algorithms) are sensitive to the input representation [10]. One way to view a representation is as the labeling of the elements of the input space (recall we are free to assign integer labels to the elements of the input space in any way we wish). The structure of the **P**-matrix immediately gives an NFLT for representations, *when we cannot make any prior assumptions about which $f\in F$ we are working on, then there is no advantage to be gained from a different representation.* Specifically, suppose we introduce another input space X', which is one-to-one related to X. Now consider the composite mapping $X'\rightarrow X\rightarrow Y$ and the associated composite function space $F'=\{f':X'\rightarrow Y\}$. The effect of the new representation X' is simply to re-index the original f's (i.e., "shuffle" the columns of the **P**-matrix). Hence, the counting lemma still applies and the above statement follows immediately.

By neighborhood structure we mean the relationship between the performance of x_i and its "neighbors" x_{i-1} and x_{i+1}. Even when discretized into a finite world, real-variable functions with properties like continuity, convexity, and differentiability naturally impose "nice" neighborhood structure on the search space in the sense that the neighbors of some x_i generally give similar performance. This can make it easy for an algorithm like hill-decent to search the solution space. Combinatorial problems, like the Traveling Salesman, on the other hand, rarely seem to have such nice neighborhood structures. It should be clear that for any specific $f\in F$, a proper labeling of the input set X can produce any desired neighborhood structure (e.g., monotonicity). Thus, one can imagine fixing the search algorithm (e.g., genetic algorithm) and searching the space of input representations for the one that works best. However, since there are many more ways to index the elements of X than there are elements of X, this problem is much larger than simple brute force search over all X.

4. Insights from the P-matrix

The **P**-matrix provides many useful insights about optimization.

[2] Of course, in practice solving stochastic optimization problems is usually much harder than solving deterministic ones since to discover the best solution we not only need to explore the solution space we also need to evaluate (estimate) the expectation of each candidate we explore. In general, efficient stochastic optimization involves a tradeoff between a *breadth* component to explore the solution space X and a *depth* component to obtain increasingly more accurate estimates of the performance of each candidate examined (see [9]).

4.1 *Conservation of Performance*

The essence of the NFLT is that all row averages in the **P**-matrix are equal. Another way to view this result is as a *conservation of average performance* law. By conservation of average performance, if a strategy gives better than average performance over some subset of problems, then there must be another subset where it performs worse than average. Conservation of *average* performance, however, does not preclude the possibility of strategies that perform well above average on a few problems, but only slightly worse than average on many others. When we know nothing about the $f \in F$ we are working on, such strategies could be useful. A simple example, however, shows that such strategies are not possible.

Consider a **P**-matrix and let the rows be strategies and the columns problems. Suppose there are three possible performance values $Y=\{0,1,2\}$, larger being better. Now pick any two rows (strategies). In order for the two to have the same average performance, but for one to have worst case performance that is only slightly less than average, the distribution of the y's in the two rows would have to be different. In particular, one row would need to have more 1's than the other. However, this is not possible, since in a **P**-matrix *every* performance value in Y must appear $|Y|^{|X|-1}$ times in each row. A consequence of this is that for any pair of strategies x_i and x_j, if x_i beats x_j badly on some subset of problems, then there *must* be another subset where the opposite is true. This argument can be rigorously extended from strategies to search algorithms.

4.2 *Prior Assumptions*

An implication of the conservation of performance is that over *subsets* of the $f \in F$ the NFLT *does not* hold. Over subsets there are performance differences between strategies. This is important because the problems we encounter in the real world are usually restricted (e.g., by the laws of physics) to subsets of the possible mappings in F. The key to moving beyond the NFLT and into practice is quantifying our assumptions the mappings we are likely to encounter, determining what structural properties these assumptions imply about the likely mappings, and choosing strategies that can efficiently exploit those structural properties (see also [10-12]). The Ricatti equation solution to the LQG problem is a shining example of this.

The usual way to represent prior knowledge/assumptions about the problem we are working on is by putting a distribution, $\wp(f)$, over the columns of the **P**-matrix. At one end of the knowledge/assumptions spectrum, we know nothing about the problem $f \in F$ we are working on. This is equivalent to a uniform distribution over the columns of the **P**-matrix (i.e., making all $f \in F$ equally likely). At the other end of the spectrum we know exactly which column (i.e., which problem f) we are working on. Knowing f, however, does not mean that we know its solution. It only means that we are certain about the structure of the problem (e.g., it is strictly convex). In practice, our knowledge/assumptions is usually a distribution that lies somewhere between these two extremes.

The NFLTs in Section 3 are for the case where $\wp(f)$ is uniform over the $f \in F$. An NFLT can also be established for the case where all *priors* $\wp(f)$ are equally likely [14]. Intuitively, this follows from the NFLTs in Section 3 since all priors equally likely is equivalent to a uniform distribution over all $f \in F$.

4.3 *Performance/Sensitivity Tradeoff*

In principle if we know $\wp(f)$ (the distribution of likely problem instances), then we can choose a row of the **P**-matrix (strategy) that has better average performance than any other. This will constitute the "optimal" strategy. Focusing only on performance, however, can lead to a design that is sensitive to spectacular failure. There are two ways this might happen. In the first case, since every $y \in Y$—the best as well as the worst—appears an equal number of times in each row, if we are ever wrong about $\wp(f)$, then the performance of our "optimal" strategy can be arbitrarily poor. In the second case, we might allow occasional spectacular failures if they are more than offset by generally spectacular performance. An example of the first case is the automobile airbag. While airbags have reduced the probability of injury for adult males, small women, children, and child seats pose a sensitivity not considered in $\wp(f)$. An example of the second case are fighter aircraft. These have very high performance, but any structural damage almost inevitably results in disaster. The only hope is that the high performance minimizes the likelihood of structural damage.

4.4 *Performance/Robustness Tradeoff*

Closely related to the performance/sensitivity tradeoff is the performance/robustness tradeoff. Robust solutions attempt to overcome the sensitivities inherent in highly optimized solutions. The tradeoff is that robust designs must generally give up some performance. Again there are two cases that call for a robust design. In the first case, we admit that our knowledge about likely problem instances $\wp(f)$ is imperfect, e.g., due to limited experience with the problem domain or known modeling error. In the second case, we might have good knowledge $\wp(f)$, but the consequences of spectacular failure, however low its probability, are just too enormous to contemplate.

In the first case, if we are unsure about $\wp(f)$ then we must relax our assumptions and select a strategy that gives good performance over a larger subset of the $f \in F$. Suppose $\wp(f)$ is uniformly distributed over some subset of columns F_1. Let the highly optimized solution, x_1^*, be the row that gives the best average performance over F_1. For the robust solution, let us choose a subset F_2 such that $F_1 \subset F_2$. Assume a uniform distribution over F_2, and let the robust solution,

x_2, be the row that gives the best average performance. Note that in general rows x_1* and x_2 are not the same. Moreover, when they are not the same and x_1* is unique, then the average performance of x_2 over F_1 is *always* worse than the average performance of x_1* over F_1. In other words, a robust solution must generally give up some performance in return for reduced sensitivity to errors in $\wp(f)$.

Regarding the second case, a highly optimized solution will generally allow spectacular failures as long as they occur with low enough probability. That is, pick any subset, F_1, of the columns in a **P**-matrix. If the number of columns in F_1 is large enough, then virtually all the performance values $y \in Y$ (including the worst) will appear in some row of the subset. Assume some $\wp(f)$ over the columns, and find the row that gives the best average performance. Again, this is the highly optimized solution x_1*. Chances are that x_1* will allow the worst possible $y \in Y$ (spectacular failure), although with low probability. Now find the row, x_2, that gives the best average, but at the same time never returns the worst possible performance. In general, $x_1^* \neq x_2$, and the average performance of x_2 is less than that of x_1*. So in this case, a robust solution that tries to lower the probability of spectacular failure generally performs less well on average than one that allows an occasional disaster.

4.5 *Random Restarts*

In search problems, it is often observed that randomly restarting the search from several new initial points can be useful in getting a good solution. The framework of the **P**-matrix provides an easy explanation for this. For a specific search algorithm, let us take a fixed number of samples starting from some specific initial point $x \in X$. This constitutes a mapping. Let us say we know very little about the problem, which means that it could be any $f \in F$. In the **P**-matrix, let the rows X represent the set of possible starting points. Let the entries of the matrix be the performances from the algorithm when applied to the (unknown) problem from initial point $x \in X$. The set of possible problems are the columns of the matrix. Now depending on the problem, some rows will return more favorable results than others. Let the probability that a row (an initial starting point) results in a good return be p. Then the probability that n random starts will result in a good return is given by $1-(1-p)^n \approx np$. An n-fold increase in success probability is achieved by n random restarts! Similar statements apply if we change the rows from representing different initial starting points to representing different strategies as in Section 3.1.

4.6 *Ordinal Optimization*

As mentioned, the size of the input space $|X|$ is typically exponential in the problem size. Hence, if there is to be any hope, a search strategy must be able to locate the optimal solution after sampling a fraction of X that grows exponentially slowly in problem size. For a problem like the Minimum Spanning Tree (MST) problem we can do it with a greedy algorithm. For the similar TSP, on the other hand, no such algorithm is known and (unless P=NP) may not exist [4]. For other problems f may be hard to evaluate (e.g., requires long simulation experiments). In either case, practical limits on time and computing budget necessitate that we must "soften our goals" from insisting on the "true optimal" solution to being satisfied with a solution that is "good enough with high probability." This is the idea behind *Ordinal Optimization* (OO) [6]. How OO works is clear when couched in terms of the **P**-matrix. Specifically, OO is based on two fundamental tenets:

- *Performance* order *is easier to determine than performance* value. The idea is that to separate good solutions from bad all we need to know is whether or not $f(x_i)>f(x_j)$, we do not need to know how much better $f(x_i)$ is than $f(x_j)$, i.e., we do not need to calculate the difference $f(x_i)-f(x_j)$. It turns out that determining performance order requires much less effort than determining performance value [3].[3] This saves computational effort. In addition, with performance *order* rather than performance *value*, the number of possible functions (columns in the **P**-matrix) is $|X|^{|X|}$ rather than $|Y|^{|X|}$. That is, with order we replace the performance value space $\{y_0,y_1,\ldots,y_{|Y|-1}\}$ with the performance order space $\{1,2,\ldots,|X|\}$, and the f map each input $\{x_0,x_1,\ldots,x_{|X|-1}\}$ to its performance order in $\{1,2,\ldots,|X|\}$. When $|Y|>|X|$ this represents an exponential reduction in the universe of possible problem instances.

- *Softening the goal decreases the computational burden.* With goal softening, we no longer insist on the solution that gives the very best possible performance. Instead, we are satisfied with any solution that is "good enough," e.g., gives performance that is ranked in the top 1%. That is, instead of seeking the solution that returns a single point (the best performance value), we accept any solution that returns performance that falls into a "good enough subset" of the performance space Y. Since each $y \in Y$ appears $|Y|^{|X|-1}$ times in each column of the **P**-matrix, each additional value of Y that we accept as "good enough" adds $|Y|^{|X|-1}$ more columns to the subset of F for which a given $x \in X$ provides an acceptable solution. In this way we have a much higher probability of success after exploring only a small fraction of the inputs.

4.7 *Optimization against Adversaries*

An alternative way to look at optimization is as a two-player matrix game with the **P**-matrix playing the role of the payoff matrix. For the optimization problems considered so far, "nature" chooses a column (problem instance) according to some $\wp(f)$, and our goal is choose the row (strategy) that optimizes our expected payoff. Imagine instead that we face an adversary who follows our

[3] To see this yourself, take two boxes, one in each hand. It is much easier to decide which box is heavier than it is to decide precisely how much heavier.

choice of strategy by deliberately trying to pick problem instances that return poor performance (good for the adversary, bad for us). This is the sort of problem faced by information technology administrators in trying to secure a computer networks against hacker attack.

Examining the **P**-matrix makes the difficulty with adversaries clear. Because each $y \in Y$ appears an equal number of times in every row, it is always possible for an adversary to pick a column (attack) that returns what from our point of view is the least desirable outcome. In other words, *against an adversary of unlimited power, all strategies have the same performance* (see also [1,2]). Here unlimited power is in the sense that the adversary has the ability to search the **P**-matrix for the column (security attack) that gives the worst performance for the row (security strategy) we have picked. It is assumed that while the adversary is probing our system for security holes, our security strategy is not changing. For computer network security this is often the situation, since it is typically only after a successful attack that the security strategy is changed.

Optimization against adversaries makes it clear that security is much harder than performance optimization. In optimization, "nature" picks its problem instances according to some generally invariant laws. In security, in contrast, our adversary is constantly learning, adapting, and discovering new attacks. Moreover, the computing power available to them to use in carrying out attacks is growing exponentially (Moore's law). On the other hand, what the **P**-matrix hides is the effort in formulating different attacks, some $f \in F$ are much more difficult to realize. Conceptually, what we try to do in security engineering is to design our system so that only those attacks that we can "defend" are achievable. For those we cannot defend we try, for example, to guarantee that it would require an exponential amount of computational effort for an attacker to realize them. This is the idea behind cryptography for instance.

5. Conclusions

In a finite world, the universe of all possible optimization problems can be summarized by a **P**-matrix. In the broadest sense, the rows of this matrix can be taken as strategies, the columns as all possible problems, and the entries as the performances of the strategies over the problems. That all rows have the same average is the NFLT in a nutshell, making it clear that *if anything is possible, then nothing can be expected*. The value of the **P**-matrix and NFLT is that they render certain assertions or conclusions either obvious or problematical, e.g., strategy x is good—not if you do not specify the class of problems you intend to apply it to. In this sense it is clear that the assumptions are the key. The practice of optimization boils down to what assumptions we can make about likely problem instances, what structural properties do the assumptions imply, what strategy is best matched that structure, and how sensitive our strategy is to the assumptions. While we have come far, our ever increasing reliance on large scale networked infrastructures (e.g., power grids and Internets), which are currently not well understood and inarguably insecure, has seemingly invalidated many classic assumptions and opened up many exciting challenges.

References

[1] Culberson, J.C., *On the Futility of Blind Search*, Technical Report TR 96-18, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, July 1996.

[2] Culberson, J.C., "On the Futility of Blind Search: An Algorithmists View of the 'No Free Lunch'", *Evolutionary Computation*, No. 6, pp. 109-127, 1998.

[3] Dai, L., "Convergence Properties of Ordinal Comparisons in the Simulation of DEDS," *JOTA*, Nov. 1996.

[4] Du, D.-Z and K.-I. Ko, *Theory of Computational Complexity*, Wiley-Interscience, 2000.

[5] Duda, R.O., P.E. Hart, and D.G. Stork, *Pattern Classification* (2nd ed.), John Wiley & Sons, 2001.

[6] Ho, Y.-C., "An Explanation of Ordinal Optimization: Soft Computing for Hard Problems," *Information Sciences*, 1999.

[7] Ho, Y.-C., "The No Free Lunch Theorem and the Human Machine Interface," *IEEE Control Systems Magazine*, June 1999.

[8] Koppen, M., D.H. Wolpert, and W.G. Macready, "Remarks on a recent paper on the 'No Free Lunch' theorems," *IEEE Transactions on Evolutionary Computation*, Vol. 5, No. 3, pp. 295-296, June 2001.

[9] Lin, X.-C., *A New Framework for Discrete Stochastic Optimization*, Ph.D. Dissertation, Harvard University, 2000.

[10] Radcliff, N.J. and P.D. Surry, "Fundamental Limitations on Search Algorithms: Evolutionary Computing in Perspective," *Computer Science Today*, 1000, pp. 275-291, 1995.

[11] Sharpe, O., "Beyond NFL: A Few Tentative Steps," available online.

[12] Sharpe, O., "Continuing Beyond NFL: Dissecting Real World Problems," available online.

[13] Wolpert, D.H. and W.G. Macready, "No Free Lunch Theorems for Optimization," *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, pp. 67-82, April 1997.

[14] Wolpert, D.H. and W.G. Macready, *No Free Lunch Theorems for Search*, Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.