JOURNAL OF OPTIMIZATION THEORY AND APPLICATIONS: Vol. 115, No. 3, pp. 549–570, December 2002 (© 2002)

Simple Explanation of the No-Free-Lunch Theorem and Its Implications[1]

Y. C. Ho[2] and D. L. Pepyne[3]

Communicated by M. A. Simaan

Abstract. The no-free-lunch theorem of optimization (NFLT) is an impossibility theorem telling us that a general-purpose, universal optimization strategy is impossible. The only way one strategy can outperform another is if it is specialized to the structure of the specific problem under consideration. Since optimization is a central human activity, an appreciation of the NFLT and its consequences is essential. In this paper, we present a framework for conceptualizing optimization that leads to a simple but rigorous explanation of the NFLT and its implications.[4]

Key Words. No-free-lunch theorem, optimization, learning, decision making, search, strategy selection, impossibility theorem, representation and encoding, robustness, sensitivity, complexity.

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

[1]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]Research Professor, Division of Engineering and Applied Sciences, Harvard University, Cambridge, Massachusetts.
[3]Research Associate, Division of Engineering and Applied Sciences, Harvard University, Cambridge, Massachusetts.
[4]An earlier version of this paper was published in the Proceedings of the IEEE Conference on Decision and Control, December 2001. Additional work on the relation between the no-free-lunch theorem, computational complexity, and network security will appear separately in another paper by Y. C. Ho and Q. C. Zhao.

ideal of a perfect democracy. The no-free-lunch theorem (NFLT, Refs. 1–10), 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 in 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 much of human activity. One might even go so far as to say that optimization—improving the human condition—is the goal of civilization. This makes 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 that 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 it does not impose any real loss of generality, since virtually all optimization nowadays is done using numerical algorithms or simulation programs running on digital computers. In a finite world, all the information about an optimization problem can be summarized in a matrix that we call the fundamental matrix F. In its most broad interpretation, the rows of F are strategies, the columns are the universe of all possible problems, and the matrix entries are the performances of the strategies on the problems. The essence of the NFLT is that the row averages of F are always equal; i.e., averaged over all possible problems, all strategies give the same performance. Deeper structure of F leads to other NFLTs and many useful insights.

The remainder of the paper is organized as follows. Section 2 gives the definitions and essential background necessary for the rest of the paper by defining the F-matrix and developing its properties. In Section 3, we use the F-matrix and its properties to prove NFLTs for optimal strategy selection, search algorithms, input space encodings, and stochastic optimization problems. Section 4 then discusses several implications of the NFLT, including the conservation of performance, conservation of robustness, representation of prior knowledge, performance/sensitivity tradeoff, performance/robustness tradeoff, and the complexity/fragility tradeoff. Section 5 looks at how the F-matrix can be used to understand optimization algorithms such as random restarts and ordinal optimization (Ref. 11) and observations about computational complexity and the challenge of optimization against malicious adversaries. The paper closes in Section 6 with a short conclusion.

2. Definitions and Essentials

Consider the mapping $y = f(x)$, where x is a candidate from a solution set X, y is a scalar from a performance set Y, and f is a cost or objective function. The optimization problem is to choose the solution $x \in X$ whose performance y is best in some sense (e.g., minimizes f). Under our finite world assumption, the sets X and Y are discrete and finite in size representing, for example, the input and performance spaces of a discretized continuous-variable optimization problem or the set of tours and discretized tour lengths of a combinatorial traveling salesman problem (TSP). The next lemma establishes that, when the sets X and Y are finite, then the universe of unique mappings from X to Y is also finite.

Lemma 2.1. If X has size $|X|$ and Y has size $|Y|$, then there are $|F| = |Y|^{|X|}$ unique mappings in the set $F = \{f: X \rightarrow Y\}$.

Proof. Each $x \in X$ may be mapped to one of $|Y|$ possible outputs. Since there are $|X|$ inputs, it follows immediately that there are at most $|F| = |Y|^{|X|}$ possible unique mappings from X to Y. □

2.1. Fundamental Matrix. Since the sets X, Y, F are all discrete, we can assign integer labels to each of their elements, i.e., let

$$X = \{x_0, x_1, \ldots, x_{|X|-1}\}, \qquad Y = \{y_0, y_1, \ldots, y_{|Y|-1}\},$$
$$F = \{f_0, f_1, \ldots, f_{|F|-1}\}.$$

How we label the elements is entirely arbitrary. The universe of all possible unique mappings from X to Y can then be summarized by a matrix that we call the fundamental matrix F.

Definition 2.1. A fundamental matrix F is a matrix with $|X|$ rows, one for each $x \in X$, $|F|$ columns, one for each $f \in F$, and ijth entry $\mathrm{F}_{ij} = f_j(x_i) \in Y$; i.e., the ijth entry is the performance of candidate solution x_i on cost function f_j.

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

$$\begin{array}{c|c|c|c|c|c|c|c|c|} & f_0 & f_1 & f_2 & f_3 & f_4 & f_5 & f_6 & f_7 \\ \hline x_0 & y_0 & y_1 & y_0 & y_1 & y_0 & y_1 & y_0 & y_1 \\ \hline x_1 & y_0 & y_0 & y_1 & y_1 & y_0 & y_0 & y_1 & y_1 \\ \hline x_2 & y_0 & y_0 & y_0 & y_0 & y_1 & y_1 & y_1 & y_1 \\ \hline \end{array} \, . \tag{1}$$

Another way to view the construction of F is to look at the matrix formed by the integer labels of the y's, which for the example above is

$$\begin{array}{c|c|c|c|c|c|c|c|c|} & f_0 & f_1 & f_2 & f_3 & f_4 & f_5 & f_6 & f_7 \\ \hline x_0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ \hline x_1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ \hline x_2 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline \end{array} \, . \tag{2}$$

As seen, the columns in (2) consist of all of the 3 bit binary integers 000 through 111. In the general case, the columns of F will consist of all the $|X|$ digit, base-$|Y|$ integers. Thus, one way to construct the F matrix is to count modulo-$|Y|$ through all of the $|X|$ digit integers. Formed in this way, the *ij*th entry of F is the index label of the element of Y at that location. This construction makes it clear that each column of a fundamental matrix F is not only unique, but that the columns exhaust the universe of all possible unique columns.

In practice, the size of the input set $|X|$ is often very large (e.g., exponential in the dimension of a discretized continuous-variable problem or combinatorial in the number of cities in a TSP). Likewise, $|Y|$ is also often huge, making $|F| = |Y|^{|X|}$ an impossibly large number. This generally precludes actually constructing and storing a fundamental matrix F. But as a conceptual and pedagogical tool, the framework of the fundamental matrix F leads to simple intuitive proofs of the NFLTs and provides many useful insights about optimization and optimization approaches.

2.2. Properties of the F Matrices. Fundamental matrices have some special structural properties, many of which come from the following observation. Inspecting the F matrix in (1) reveals that y_0 and y_1 appear the same number of times (four) in every row. This key result is formally established in the following fundamental counting lemma.

Lemma 2.2. Counting Lemma. Consider the fundamental matrix F associated with some given X of size $|X|$ and given Y of size $|Y|$. In this matrix, each $y \in Y$ appears $|Y|^{|X|-1}$ times in each row.

Proof. By Lemma 2.1, the F-matrix has $|Y|^{|X|}$ columns. Thus, if each $y \in Y$ appears an equal number of times in each row, then the number of times each y must appear is $|Y|^{|X|}/|Y| = |Y|^{|X|-1}$. Accordingly, if some $y \in Y$ appears less than $|Y|^{|X|-1}$ in some row, then some other $y \in Y$ must appear more than $|Y|^{|X|-1}$ times in that same row. With this in mind pick any row i and any $y \in Y$. Now, form a submatrix by eliminating from F all columns for which the matrix entry $F_{ij} \neq y$. Since all columns of an F-matrix are unique, it follows that the columns of this submatrix must also be unique. Moreover, since all of the columns of the submatrix share a common value of y in row i, it follows that, when row i is eliminated, the columns of the submatrix must still be unique. But given that this submatrix has $|X| - 1$ rows and there are $|Y|$ possible outputs, it follows (from Lemma 2.1) that there can be no more than $|Y|^{|X|-1}$ columns in this submatrix if its columns are to be unique. Uniqueness of the columns of F therefore requires that each $y \in Y$ appear exactly $|Y|^{|X|-1}$ times in each row. $\square$

The next corollary follows directly from the counting lemma.

Corollary 2.1. For a fundamental matrix F, the row sums (hence row averages) are all equal.

Proof. Since each $y \in Y$ appears $|Y|^{|X|-1}$ times in each row, the row sum for any row i is

$$\sum_{j=0}^{F-1} \mathrm{F}_{ij} = \sum_{k=0}^{Y-1} y_k |Y|^{|X|-1} \tag{3}$$

and the row average is

$$(1/|F|) \sum_{j=0}^{F-1} \mathrm{F}_{ij} = (1/|Y|^{|X|}) \sum_{k=0}^{Y-1} y_k |Y|^{|X|-1} = (1/|Y|) \sum_{k=0}^{Y-1} y_k. \tag{4}$$

□

The next result establishes that, if we successively eliminate specific columns and rows from a fundamental matrix, the resulting submatrices are themselves fundamental matrices.

Corollary 2.2. Consider the fundamental matrix F associated with some given X of size $|X|$ and given Y of size $|Y|$. Pick any row i and any $y \in Y$. The submatrix formed by eliminating from F row i and all columns j such that $\mathrm{F}_{ij} \neq y$ has $|X|-1$ rows and $|Y|^{|X|-1}$ columns is an F-matrix; in particular, it is the F-matrix associated with the input set $X = X \backslash x_i$ (the original input set X with x_i removed) and output set Y.

Proof. Before proving this result, let us illustrate. To do so, begin with the fundamental matrix in (1). Now, suppose that we eliminate row x_0 and all columns j such that $f_j(x_0) \neq y_0$, i.e., eliminate from (1) columns f_1, f_3, f_5, f_7. The resulting submatrix is

$$\begin{array}{c|c|c|c|c|} & f_0 & f_2 & f_4 & f_6 \\ \hline x_1 & y_0 & y_1 & y_0 & y_1 \\ \hline x_2 & y_0 & y_0 & y_1 & y_1 \\ \hline \end{array}. \tag{5}$$

As can be seen, this matrix is itself the F-matrix for $X = \{x_1, x_2\}$ and $Y = \{y_0, y_1\}$, and hence satisfies Lemmas 2.1, 2.2, and Corollary 2.1. That is, all columns of the submatrix are unique, each $y \in Y$ appears an equal number of times in each row, and its row sums/averages are all the same. Moreover, we can easily see that the submatrix also satisfies Corollary 2.2.

In particular, if we now eliminate x_2 and all columns for which $f_j(x_2) \neq y_0$, we get the F-matrix

$$\begin{array}{c|c|c|} & f_0 & f_2 \\ \hline x_1 & y_0 & y_1 \\ \hline \end{array}. \tag{6}$$

To formally prove the result, it is clear that the submatrix will have $|X|-1$ rows and $|Y|^{|X|-1}$ columns, since by Lemma 2.2 each $y \in Y$ appears $|Y|^{|X|-1}$ times in each column of the original F-matrix. All that remains is to show that the submatrix is a fundamental matrix. Doing this requires showing that its columns are unique. Recalling the proof of Lemma 2.2, this follows immediately from the fact that the columns of the original F-matrix from which the submatrix was formed are unique. $\square$

While other properties can be established for the fundamental matrices, those established above are sufficient for our purposes in this paper, and we are now in a position to present the no-free-lunch theorems of optimization.

3. No-Free-Lunch Theorems

The fundamental matrix, the counting lemma, and its corollaries established in Section 2 are at the heart of simple and intuitive proofs and explanations of the NFLTs of optimization. The essential idea is the following. Let x be an optimization variable, f a performance function, and $y=f(x)$ the performance of x on f. Assume a finite world and construct the associated fundamental matrix F. Then, averaged over all possible performance functions (columns of F), the performance is choice (row of F) independent (since all row averages of F are equal). In other words, "no choice is universally better than any other." This is the NFLT in a nutshell. Different interpretations of the elements of sets X and Y and utilization of the properties of the fundamental matrices lead to more specific NFLTs, some of which we develop next.

3.1. Optimal Strategy Selection. In its broadest interpretation, the x's (rows of F) are strategies, the f's (columns of F) are all possible optimization problems, and $f(x) \in Y$ is the performance of strategy x on problem f. In this context, a strategy is a mapping from the space of available information to a control variable or decision space. Strategies include methods and algorithms involving search, adaptation, learning, voting, feedback, dynamic programming, evolution, randomization, auctions, games, and even humans in the loop. In short, the concept of strategy is any process that converts available information into decisions. Nothing can be more general or more inclusive.

For a finite information space I of size $|I|$ and a finite decision space D of size $|D|$, there are $|D|^{|I|}$ possible strategies, typically a huge number, but still finite. Thus, if we let X be a strategy space, then

$$|X| = |D|^{|I|},$$

in which case there are

$$|F| = |Y|^{|X|} = |Y|^{|D|^{|I|}}$$

possible problems! In this case, the fundamental matrix is impossibly huge for all but the simplest toy problems. Nevertheless, we know that its row averages are all equal, which leads immediately to the conclusion that

> "there is no strategy of any kind that outperforms all others on all problems."

In this sense, the NFLT is an impossibility theorem suggesting that

> "universal optimizers are impossible."

3.2. Search Algorithms. A more specific and familiar problem involves function optimization, e.g.,

$$\min_{x \in X} f(x), \tag{7}$$

where the x's are vectors of input parameters, the y's are the costs associated with the different input vectors, and the f's are different (deterministic) cost functions. Typically, we solve such problems by searching the space X for the element that give the best (e.g., least) cost. Now, since a search algorithm is simply a strategy for solving an optimization problem, we can immediately conclude from Section 3.1 that[5]

> "there can be no search algorithm that outperforms all others on all problems."

But this does not provide any intuition. A proof that explicitly considers the search process is much more insightful.

[5]In the context of Section 3.1, since the number of search algorithms that run in finite time on a digital computer with finite memory is finite, we take each search algorithm as a different strategy (row of F), each column as a different optimization problem, and each element of Y as the performance of algorithm x on optimization problem f.

Such a proof for search algorithms that stop after picking m distinct samples from X was developed by Wolpert and Macready (see Refs. 5, 9, 10). Their proof is based on statistical arguments, and is complicated by the fact that it must be established for all possible search algorithms, including those that learn and adapt based on the performances which they observe as they sample. The concept of the fundamental matrix leads to an alternative proof that we believe is more straightforward and intuitive.

In order to compare search algorithm performance, two things are required. First,

"the algorithms must use the same information."

In Wolpert and Macready, this is the requirement that the search algorithm halt after collecting $m \leq |X|$ distinct samples of the x's in X. Each sample improves the chances of getting a good solution, so algorithms can only be fairly compared if they take the same number of samples. Moreover, since the f's are assumed deterministic, nothing is gained by resampling a previous x, so the samples collected should be distinct. Second,

"the algorithms must use the information rationally."

This means that, if the goal is minimization, the algorithm must return the sample x (from the m samples collected) that gives the smallest cost $f(x)$. No other choice makes rational sense.

With the above in mind, assume that the sets X and Y are discrete and finite, in which case the set $F = \{f{:}X \rightarrow Y\}$ is also discrete and finite. Construct the fundamental matrix, letting the rows be the elements of X and the columns the elements of F. Following Wolpert and Macready, assume that we have no prior knowledge about which cost function (column) $f \in F$ we are working on—it could equally likely be any one of them. In this case, the maximal amount of information available to guide a search algorithm is the history of the x's sampled and their associated cost values $f(x)$. By sampling different $x \in X$ and observing their costs, a search algorithm tries to learn about which $f \in F$ it is working on so as to locate the optimum with as few (distinct) samples as possible. Under these assumptions, Corollaries 2.1 and 2.2 can be used to rigorously establish that

"no search algorithm, no matter how sophisticated, should a priori be expected to give better performance than any other."

We illustrate the idea of the proof via a simple example. Specifically, let $X = \{x_0, x_1, x_2\}$ and $Y = \{0, 1\}$ (e.g., $0 =$ good, $1 =$ bad), in which case the F matrix is

$$\begin{array}{c|c|c|c|c|c|c|c|c|} & f_0 & f_1 & f_2 & f_3 & f_4 & f_5 & f_6 & f_7 \\ \hline x_0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ \hline x_1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ \hline x_2 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline \end{array}. \tag{8}$$

Now, if we knew which column of (8) we were working on, then an algorithm could be developed that immediately returns the optimal row. But, since we assume no prior knowledge about the column we are working on, we can only sample the space X to try to determine the column and the optimal row. Let us consider the performance we can expect after each sample.

Sample $m = 1$. When all columns are equally likely, then clearly no matter how a search algorithm picks its first sample, the expected performance is the same since all row averages are equal (Corollary 1).

Sample $m = 2$. Now, after sampling some x_i and observing its cost $y = f(x_i)$, some information is gained and we can eliminate certain columns from further consideration. In particular, suppose that we sample x_0 and observe the cost $f(x_0) = 1$ (then, unless we are working on column f_7, we have not found the optimal and must continue searching). Eliminating from (8) row x_0 and all columns f_j such that $f_j(x_0) \neq 1$ gives[6]

$$\begin{array}{c|c|c|c|c|} & f_1 & f_3 & f_5 & f_7 \\ \hline x_1 & 0 & 1 & 0 & 1 \\ \hline x_2 & 0 & 0 & 1 & 1 \\ \hline \end{array}. \tag{9}$$

But the submatrix that remains after elimination is itself an F matrix (Corollary 2.2). Moreover, by our assumption of no prior knowledge, the column that we are working on is likely to be any of those that remain. Hence, rows x_1 and x_2 have the same expected performance (Corollary 2.1). In other words, while a history of previously sampled x's and their associated costs $f(x)$ does provide information about which columns we are not working on, the history does not help to tell us which of those that remain we are working on. No information is gained from the sampling process that can help guide the search process, and the expected performance of the next sample is the same regardless of how that sample is generated.[7]

[6]In general, after each sample, we can eliminate from further consideration $|Y|^{|X|-1}$ columns (Lemma 2.2).

[7]This includes deterministic and stochastic sampling schemes.

In general, after $0 \leq m \leq |X|$ samples from X are evaluated, if one more distinct sample x is allowed, the cost $f(x)$ associated with this new sample, no matter how sophisticated the scheme used to generate it, is the same on average. Consequently,

> "absent any prior assumptions about the cost function being worked on, all search algorithms have the same expected performance." □

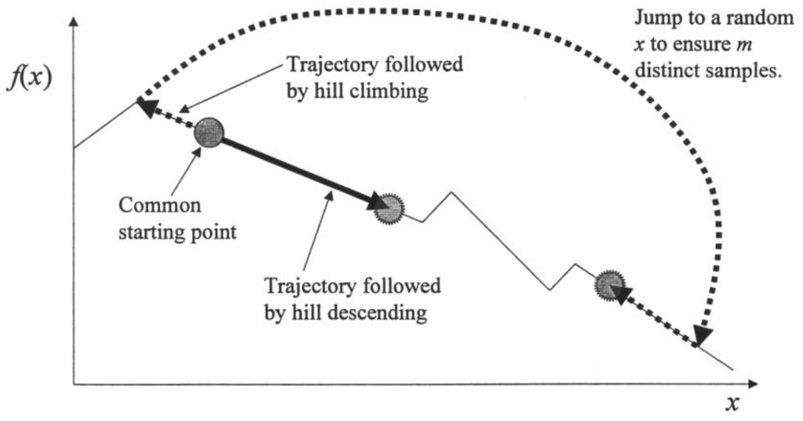
The NFLT for search leads to some interesting and counterintuitive conclusions (see also Refs. 5, 9, 10). One interesting conclusion is that, unless we can make some prior assumptions about the cost function we are working on, no search algorithm can a priori be expected to perform any better than blind random pick. The risk that we take is that our algorithm might actually perform worse than random on the specific problem we are trying to solve! This is generally unacceptable since blind random pick is particularly inefficient. In particular, assuming that all $f \in F$ are equally likely, then since each $y \in Y$ appears an equal number of times in each row (Lemma 2.2), the probability that any sample $x \in X$ returns the optimal $y \in Y$ is $p = 1/|Y|$. Thus, the probability of finding the optimal after n independent random samples is given by the geometric distribution, $p(1-p)^{n-1}$, in which case the expected number of random samples needed to locate the optimal is $1/p = |Y|$. Since $|Y|$ is generally very large, the number of samples required by random search can be enormous. For example, if we discretize the output space to only 32 bits (a relatively crude discretization), then on average random search requires $|Y| = 2^{32} \approx 10^9$ samples to find the optimal.[8]

Another way of stating the NFLT for search is that

> "averaged over all cost functions, all search algorithms give the same performance."

A counterintuitive conclusion of this is that on average hill climbing performs the same as hill descending, even when the goal is function minimization! Hill climbing in a finite one-dimensional input space would compare $f(x_i)$ to $f(x_{i+1})$ and proceed to x_{i+1} if $f(x_i) \leq f(x_{i+1})$; otherwise, it would proceed to x_{i-1}. Hill descending would do exactly the opposite. In both cases some rule would be needed to ensure the x's are distinct, e.g., when the algorithm reaches a local maximum (in the case of hill climbing) or a

[8]Here, we have assumed sampling with replacement, i.e., the x's are not required to be distinct. Even for the case where the x's are distinct, sampling with replacement is a reasonable assumption when $n \ll |X|$. When n is on the order of $|X|$, the probability of finding the optimal after n distinct samples depends both on $|Y|$ and on $|X|$.



$f(x)$
Trajectory followed by hill climbing
Jump to a random x to ensure m distinct samples.
Common starting point
Trajectory followed by hill descending
x

Fig. 1. Example where hill climbing outperforms hill descending on a minimization problem.

local minimum (in the case of hill descending), some scheme would be needed to jump to an x that has not already been tried. Suppose that the two algorithms always begin from the same $x \in X$ and use exactly the same scheme for ensuring distinct x's. In this case, then clearly on monotonic functions [i.e., those such that $f(x_i) \leq f(x_{i+1})$ for all $i = 0, 1, \ldots, |X| - 1$], hill descending would find better solutions. For flat functions [i.e., $f(x_i) = f(x_{i+1})$], the two would give equal performance. But over many jagged functions, hill climbing may actually find better solutions than hill descending, such as for the example in Fig. 1.[9] Notice in the figure that we have tried to make the total distance traversed by the two search algorithms the same to suggest that they stop after the same number of distinct samples.

3.3. Encodings and Neighborhood Structures. It is well known that the performance of certain search algorithms (e.g., genetic algorithms) are sensitive to the encoding or representation (Ref. 6). One way to view an encoding is as a specific indexing of the elements of the input space X (recall that we are free to assign integer labels to the elements of the input space in any way we wish). Different encodings lead to different neighborhood structures. Neighborhood structure describes the relationship between the cost of x_i and its neighbors x_{i-1} and x_{i+1} in the one-dimensional case. Even when discretized into a finite world, real-variable functions with properties like

[9]Recall that a search algorithm has two parts: the part that chooses the next distinct sample and the part that returns the sample giving the best cost value. Here, we are assuming only that the sampling part is different; both algorithms return the sample giving the minimum cost value.

continuity, convexity, and differentiability naturally impose nice neighborhood structure on the search space in the sense that the neighbors of x_i generally give similar cost. This can make it easy for an algorithm like hill-descent to search the solution space. Combinatorial problems, like the traveling salesman problem, on the other hand, rarely seem to have such nice neighborhood structures. In any case, it should be clear that, for any specific mapping $f \in F$, a proper labeling of the elements of the input set X can produce any desired neighborhood structure (e.g., convexity or monotonicity). Thus, for a given search algorithm, like hill descent, different encodings will give different performance. One can imagine then fixing the search algorithm and looking for the encoding that gives the neighborhood structure that is best matched to the search algorithm. However, since there are many more ways to index the elements of X than there are elements of X ($|X|!$ vs. $|X|$), searching the space of possible encodings is an even a more daunting task than exhaustive enumeration of the entire input space. Moreover, using the properties of fundamental matrices we can immediately establish the following lemma.

Lemma 3.1. NFLT for Encodings. For a given search algorithm, and absent any prior information about which $f \in F$ we are working on, no encoding can be expected to result in better performance than any other.

Proof. Suppose that we introduce another input space X', which is one-to-one related to X. Now, consider the composite mapping $X' \to X \to Y$ and the associated composite function space $F = \{f' : X' \to Y\}$. The effect of the new encoding X' is simply to reindex the original f's (i.e., interchange the columns of the F-matrix). Hence, the counting lemma (Lemma 2.2) still applies and the result follows immediately. □

3.4. Stochastic Optimization. Sometimes, we cannot evaluate the cost function in (7) exactly, but rather we have $z = l(x,\omega)$, where x is a candidate solution, z is a cost value, and $\omega \in \Omega$ is a random quantity reflecting error in our ability to evaluate the x's performance. This leads to the stochastic optimization problem,

$$\min_{x \in X} E_\Omega[l(x, \omega)], \tag{10}$$

where E is the expectation operator. Now, if we let

$$y = f(x) = E_\Omega[l(x, \omega)],$$

where $x \in X$ and $y \in Y$, then conceptually a stochastic optimization problem is identical to a deterministic one.[10] Consequently, we can appeal to our previous results to say that

> "there is no universal strategy for stochastic optimization problems."

That is, if we know nothing about the function l or the distribution Ω, then we should not expect one strategy to outperform any other.

4. Insights from the F-Matrix

The framework of the fundamental matrix can be used to explain certain laws and limitations encountered in solving optimization problems.

4.1. Conservation of Performance. Consider an F-matrix whose rows are strategies and whose columns are problems. According to Corollary 2.1, if a strategy gives the better average performance over some subset of problems (columns of **F**), then there must be another subset of problems where its performance is worse than average. In other words,

> "average performance is always conserved."

Conservation of average performance, however, does not preclude the possibility of a strategy that performs well above average on some problems and only slightly below average on the rest. Such strategy would be desirable when we know little about the problem (column) we are working on and want to minimize the probability of bad performance. What rules out this strategy is the counting lemma (Lemma 2.2), because every performance value (the very best as well as the very worst) must appear exactly the same number of times in each row of F. In fact, the counting lemma leads immediately to the following performance conservation law.

Lemma 4.1. Conservation of Performance. 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 equally large subset where the opposite is true. This holds

[10]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 input space X but we also need to evaluate (estimate) the expectation of each candidate $x \in X$ we explore. In general, efficient stochastic optimization involves a tradeoff between a breadth component to explore X and a *depth* component to obtain increasingly more accurate estimates of the expectation of the performance of the candidates examined (see Ref. 12).

regardless of the criteria used to compare the strategies (e.g., average cost, best observed cost, worst observed cost, etc.).

4.2. Conservation of Robustness. Closely related to performance conservation is the conservation of robustness. In an engineering sense, a strategy is robust if it guarantees a certain level of performance over a range of problems. For instance, suppose that,

$$Y=\begin{cases}1, & \text{good performance,}\\ 0, & \text{bad performance.}\end{cases}$$

Then, by the counting lemma (Lemma 2.2), all strategies give good performance for $|Y|^{|X|-1}/|Y|^{|X|}=1/|Y|$ of the problems (columns of F). For the example above, all strategies give good performance over 1/2 of the problems and bad performance over the other half. Every strategy is robust and yet fragile in this way, and we have the following lemma.

Lemma 4.2. Conservation of Robustness. No strategy is universally more robust than any other.

4.3. Prior Knowledge/Assumptions. The problems encountered in practice are usually restricted (e.g., by the laws of physics) to subsets of the columns of F. Over subsets of the columns, the row sums are not generally equal, in which case some row choices (strategies) will give better performance than others. Thus, if something is known about which column f comes from, the choice of row can be specialized to this knowledge.

Conceptually, our prior knowledge/assumptions about the problem we are working on can be expressed as a distribution over the columns of F. 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 F (making all $f\in F$ equally likely). At the other end of the spectrum, we know exactly which column f we are working on.[11] In practice, our knowledge/assumptions is usually a distribution that lies somewhere between these two extremes.[12]

[11]Knowing f, however, does not mean that we know its solution (we may still have to search over the rows of F). It only means that we are certain about the structure of the problem (e.g., it is strictly convex).

[12]The NFLTs in Sections 3.1–3.3 are for the case where the distribution is uniform over the columns of F. The NFLT in Section 3.4 includes the case where all priors over the columns of F are equally likely.

4.4. Performance/Sensitivity Tradeoff. With prior knowledge/assumptions expressed as a distribution over the columns of F, our goal is to choose the strategy (row of F) that optimizes the expected performance over this assumed distribution. The optimal strategy is clearly one that concentrates good outcomes under the high probability columns and bad outcomes under the low probability columns.

Focusing only on optimizing performance, however, can lead to a solution that is sensitive to catastrophe. Intuitively, there are two ways this can 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 (Lemma 2.2), if our knowledge/assumptions about the distribution over the columns is ever wrong, then the performance of our optimal solution can be arbitrarily poor. In the second case, the solution giving the best expected performance for the assumed distribution over the columns may allow occasional catastrophes as long as they are more than offset by generally good performance. Specifically, since each outcome $y \in Y$ appears an equal number of times in each row, there may not be a row that can assign every bad outcome to a low probability column (in particular, if the assumed distribution gives nonzero probability to a very large fraction of the columns). 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 by the original design assumptions. Examples of the second case are fighter aircraft. These give high performance over a wide range of flight conditions, but any structural damage almost inevitably results in disaster. It is hard to choose a design that has both high performance and robustness against failures. The only hope is that the high performance minimizes the likelihood of structural damage.

4.5. Performance/Robustness Tradeoff. As described above, highly optimized designs can be very sensitive to the assumed distribution over the columns of F. A robust design attempts to overcome this sensitivity. The tradeoff is that a robust design must generally give up some performance. To illustrate, suppose that our prior assumptions give equal probability to every column in some subset F_1. Now, if we are unsure about our prior assumptions, we should be conservative and select a strategy that gives good performance over a larger subset of the columns F_2, where $F_1 \subset F_2$. Let the highly optimized solution x_1 be the row that gives the best average performance over F_1. Assume a uniform distribution over F_2, and let the robust solution x_2 be the row that gives the best average performance over F_2. Note that the rows x_1 and x_2 are not generally the same. Moreover, if rows x_1 and x_2 are not the same and they are both unique, then the performance of x_1 over F_1 is always better than the performance of x_2 over F_1. In other words,

> "a robust solution must generally give up some performance in return for reduced sensitivity to errors in the prior knowledge/assumptions."

4.6. Complexity/Fragility Tradeoff. There appears to be a fundamental tradeoff between the complexity of a design and its fragility (i.e., its sensitivity to catastrophic failure). In the context of the F-matrix, this can be understood as follows. Conceptually, the complexity of a design can be viewed as an increase in the number of design choices $|X|$. In choosing our final design (row of F) only a polynomial amount of computation is possible, which means that we can only optimize against a polynomial number of the columns. These are the planned for columns. Now, as the number of design choices increases, the number of times the best performance value $y \in Y$ appears in each row increases exponentially fast according to $|Y|^{|X|-1}$ (Lemma 2.2). As a result, as design complexity increases, the possibility that there exists a solution $x \in X$ that gives the best performance for all of the planned for columns rapidly increases. Thus, assuming that the number of columns that we have to optimize against grows slowly with increasing design complexity (slower than $|Y|^{|X|-1}$), then up to a certain point, increasing complexity can result in improved performance.

The downside of increased design complexity is that it can make a system increasing fragile and sensitive to failure (poor performance). The idea here is this. Because the total number of columns $|F| = |Y|^{|X|}$ increases exponentially fast as design complexity increases, the probability that a design will face an unplanned for column also rapidly increases. Moreover, since our design will try to concentrate bad performances under the unplanned for columns, the occurrence of any unplanned for column will tend to give bad performance. Consequently, as design complexity continues to increase, the probability of catastrophic bad outcomes increases. There is no avoiding this; systems become increasingly fragile as their complexity increases.

5. Applications

The framework of the fundamental matrix can also be used to explain the behavior of certain optimization techniques and also gives some limited insights into computational complexity.

5.1. Random Restarts. In search problems, it is often observed that randomly restarting the search from several randomly selected initial points can be a useful way to improve an algorithm's performance. The fundamental 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 $x \in X$. Let us say that we know very little about the cost function we are working on, which means that it could be any $f \in F$. In the F-matrix, let the rows X represent the set of possible starting points. The set of possible cost functions are the columns of the matrix. Let the entries of the matrix be the performance of the algorithm when applied to the (unknown) cost function from the initial point $x \in X$. Now, depending on the problem $f \in F$, 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 restarts 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 (i.e., random application of several different solution strategies improves success probability).

5.2. Ordinal Optimization. As mentioned previously, many search problems have an input space whose size $|X|$ is exponential in the problem size (e.g., the dimension of a discretized real-variable problem or the number of cities in a TSP). For other problems, the search space may not be so large, but evaluating performances $f(x)$ can be very computationally intensive (e.g., requiring long simulation runs). For the very hardest problems, $|X|$ is huge and $f(x)$ is hard to evaluate. For such problems, practical limits on time and computing budget may necessitate that we soften our goals from an insistence on the best for sure to being satisfied with a solution that is good enough with high probability. This is the essential idea behind Ordinal Optimization (Ref. 11). How, OO works is clear when couched in terms of the F-matrix. Specifically, OO is based on two fundamental tenets:

5.2.1. Goal Softening Decreases the Computational Burden. With goal softening, we no longer insist that our solution $x \in X$ give the very best $y \in Y$. Instead, we accept as good enough any solution that gives performance that ranks among the top $n\%$ of the performances in Y. Since each $y \in Y$ appears $|Y|^{|X|-1}$ times in each row of F (Lemma 2.1), each additional value of Y that we accept as good enough adds $|Y|^{|X|-1}$ more columns (problems) for which a given $x \in X$ provides an acceptable solution. In this way, the number of samples needed to guarantee a certain success probability is greatly reduced. In particular, suppose that we have no prior knowledge about which $f \in F$ we are working on, in which case we must assume that each $f \in F$ is equally likely. When all $f \in F$ are equally likely, then each $x \in X$ has a $1/|Y|$ probability of returning any specific $y \in Y$. Suppose $|Y| = 10^6$ and consider a search algorithm that stops after collecting $m = 100$ independent samples. Then, the probability that this search algorithm finds a solution that gives the best $y \in Y$ is $1 - (1 - 1/|Y|)^m = 0.0001$, whereas the probability that the search algorithm finds a solution that gives a performance ranked among the top 10% is $1 - (1 - 0.1)^m = 1$. Thus, with goal softening, we go from virtual failure to virtual certainty!

5.2.2. 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 generally requires much less computational effort than determining performance value (Ref. 13).[13] Moreover, with performance order rather than performance value, the number of possible functions (columns in the F-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.

5.3. Optimization against Adversaries. Another way to look at optimization is as a two-player, zero-sum matrix game with the F-matrix playing the role of the payoff matrix. So far, we have assumed that nature is choosing the columns (problem instances) according to some assumed stationary distribution over the columns and our goal is to choose the row (strategy) that optimizes the expected payoff. Now, imagine instead that we face an adversary who follows our choice of strategy by deliberately trying to pick a problem instance (column) that returns the worst possible payoff (bad for us, good for the adversary). This is precisely the sort of challenge faced by information technology administrators in trying to secure a computer networks against cyberattack.

Examining the F-matrix makes clear why optimization against adversaries is particularly hard. Because each $y \in Y$ appears an equal number of times in every row (counting lemma), it is always possible for an adversary to pick a column (cyberattack) that returns what from our point of view is

[13]For example, take two boxes. It is much easier to decide which box is heavier than it is to decide precisely how much heavier.

the least desirable outcome.[14] In other words,

> "against an adversary of unlimited power, all defense strategies have the same (poor) performance";

see also Refs. 1 and 2. Here, unlimited power is in the sense that the adversary knows which row (security strategy) we have chosen and has the ability to search the F-matrix for the column (cyberattack) that gives the smallest payoff. It is assumed that, while the adversary is probing our system for security holes (searching the columns), the security strategy is not changing. For computer network security, this is often the case, since a security strategy is typically only changed after a successful attack has already taken place.

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, the adversary is constantly learning, adapting, and discovering new attacks. Moreover, the computing power available to use in carrying out attacks is growing exponentially (Moore's law). On the other hand, what the F-matrix hides is the effort in formulating different attacks; some attacks (columns of F) may be more difficult to realize than others. Conceptually, what we try to do in security engineering is to design our system so that only those attacks (columns) we can defend are achievable. For those we cannot defend, we try for example to guarantee that it would require a prohibitive amount of computational effort for an attacker to realize them. This is the idea behind cryptography for instance.

5.4. Computational Complexity. The traveling salesman problem (TSP) is known to be NP-hard (cf. Ref. 14). This means that there is no known algorithm that can solve arbitrary instances of TSP problems in an amount of time that is polynomial in the number of cities to be included in the tour. A long-standing open problem in computational complexity theory is whether or not a polynomial time algorithm can ever be found for the TSP. By letting X be the set of possible tours (for N cities, there are $N!$ of these) and by letting Y be the set of possible tour lengths, the TSP can be cast in the framework of the F-matrix.[15] Now, it should be clear that, if all columns of F represent feasible TSP problem instances, then in the worst case any search algorithm will have to examine all rows $x \in X$ in order to be

[14]A well-known fact in zero-sum two-person game theory.

[15]When representing a TSP on a digital computer, the number of possible city configurations and hence the number of possible tour lengths $|Y|$ is discrete and finite.

sure of finding the optimal tour.[16] Consequently, if all columns of F are feasible, the TSP cannot have a polynomial time solution algorithm.

While the TSP does not have a known polynomial-time algorithm, the similar minimum spanning tree (MST) problem does (Ref. 14). If we let X be the set of possible spanning trees [for a graph with N vertices there are $(N\text{-}1)!$ of these] and if we let Y be the set of possible tree lengths, the MST can also be put into the framework of the F-matrix. Now, since the MST does have a polynomial-time algorithm, it follows that all columns of its F-matrix cannot be feasible. Only a subset of the columns are feasible, and the structure of the mappings represented in those columns is such that they can all be solved with a polynomial time algorithm.

Returning to the TSP, even if all columns are not feasible, that still does not imply the existence of a polynomial-time algorithm as the following "needle in a haystack" example illustrates.

$$\begin{array}{c|c|c|c|} & f_3 & f_5 & f_6 \\ \hline x_1 & 1 & 1 & 0 \\ \hline x_2 & 1 & 0 & 1 \\ \hline x_3 & 0 & 1 & 1 \\ \hline \end{array}\,. \tag{11}$$

Clearly, if all columns in (11) are possible problem instances, then in the worst case it would be necessary to check all rows in order to locate the minimizer. If such a set of columns exists for every $|X|$, then there cannot be a polynomial-time solution algorithm.

While the framework of the F-matrix is powerful for making statements and observations about what is and is not possible over the universe of all possible problem instances, it provides only limited insights about what one can expect over subsets of the problem instances. Since computational complexity involves subsets, the framework of the F-matrix is of little help; see Refs. 1 and 2 for more about the relationship between the NFLT and computational complexity.

6. Conclusions

In this paper, we developed what we believe is a pedagogically appealing framework for understanding the NFLTs of optimization and their implications. Central to our presentation is the finite world assumption in

[16]If the problem we are working on can be any of the columns of F, then the optimal cost value may be associated with any row. Thus, in the worst case, we will have to examine every row in order to locate the one giving the optimal cost.

which all input and output spaces are discrete and finite in size. This assumption does not impose any serious loss of generality, since most difficult optimization problems demand computerized assistance and digital computers with finite storage can only operate on discrete and finite sets. In a finite world, the universe of all possible optimization problems can be summarized by a matrix that we called the fundamental matrix F. In the most general setting, the rows of F are taken as strategies, the columns as all possible problems, and the entries as the performances of the strategies on the problems. That all rows of a fundamental matrix 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 NFLTs is that they render certain assertions and conclusions either obvious or problematical. For instance, the statement "strategy x is good" is problematical if you do not specify the class of problems you intend to apply the strategy to. Over subsets of the columns of F, the NFLTs do not hold, and there are generally performance differences between strategies. Thus, the assumptions one can make about the class of likely problem instances are most important. Unfortunately, general theories about the nature of likely subsets are lacking. Such theories seem to be needed to answer questions about the existence of practical algorithms (e.g., whether or not the TSP has a polynomial-time solution strategy). Without such theories, the practice of optimization will continue to be problem driven, involving quantifying our assumptions about the subsets of mappings that are likely to be encountered, determining what structural properties these likely mappings have, and choosing strategies that efficiently exploit the structural properties (see also Refs. 6, 7, 8). The Riccati equation solution to linear quadratic optimization problems is a shining example of this.

In closing, we remark that, while this paper focused on the NFLTs for optimization, the framework of the F-matrix can also be used to understand similar NFLTs for learning (cf. Ref. 3). In both cases, we sample x (rows of F) and observe $y = f(x)$ in order to determine which mapping $f \in F$ we are working on. In optimization, this information is used to predict the x^* that optimizes the given f. In learning, we use the information to build a surrogate for f, i.e., predict y for any given x. We leave it to the interested reader to generalize the results developed in this paper to the learning problem.

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, 1996.

2. Culberson, J. C., *On the Futility of Blind Search: An Algorithmic View of 'No Free Lunch'*, Evolutionary Computation, Vol. 6, pp. 109–127, 1998.
3. Duda, R. O., Hart, P. E., and Stork, D. G., *Pattern Classification*, 2nd Edition, John Wiley and Sons, New York, NY, 2001.
4. Ho, Y. C., *The No-Free-Lunch Theorem and the Human Machine Interface*, IEEE Control Systems Magazine, Vol. 19, pp. 88–90, 1999.
5. Koppen, M., Wolpert, D. H., and MacReady, W. G., *Remarks on a Recent Paper on the No-Free-Lunch Theorems*, IEEE Transactions on Evolutionary Computation, Vol. 5, pp. 295–296, 2001.
6. Radcliff, N.J., and Surry, P. D., *Fundamental Limitations on Search Algorithms: Evolutionary Computing in Perspective*, Lecture Notes in Computer Science, Springer Verlag, New York, NY, Vol. 1000, pp. 275–291, 1995.
7. Sharpe, O., *Beyond NFL: A Few Tentative Steps* (available online at http://www.cogs.susx.ac.uk/users/olivers/).
8. Sharpe, O., *Continuing Beyond NFL: Dissecting Real World Problems* (available online at http://www.cogs.susx.ac.uk/users/olivers/).
9. Wolpert, D. H., and MacReady, W. G., *No-Free-Lunch Theorems for Optimization*, IEEE Transactions on Evolutionary Computation, Vol. 1, pp. 67–82, 1997.
10. Wolpert, D. H., and MacReady, W. G., *No-Free-Lunch Theorems for Search*, Technical Report SFI-TR-95-02-010, Santa Fe Institute, Santa Fe, New Mexico, 1995.
11. Ho, Y. C., *An Explanation of Ordinal Optimization: Soft Computing for Hard Problems*, Information Sciences, Vol. 113, pp. 169–192, 1999.
12. Lin, X. C., *A New Framework for Discrete Stochastic Optimization*, Doctoral Dissertation, Harvard University, 2000.
13. Dai, L., *Convergence Properties of Ordinal Comparisons in the Simulation of DEDS*, Journal of Optimization Theory and Applications, Vol. 91, pp. 363–388, 1996.
14. Du, D. Z and Ko, K. I., *Theory of Computational Complexity*, Wiley-Interscience, New York, NY, 2000.