Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

Generalization of the No-Free-Lunch Theorem

Albert Y.S. Lam and Victor O.K. Li
Department of Electrical and Electronic Engineering
The University of Hong Kong
Hong Kong, China
{ayslam, vli}@eee.hku.hk

***Abstract*— The No-Free-Lunch (NFL) Theorem provides a fundamental limit governing all optimization/search algorithms and has successfully drawn attention to theoretical foundation of optimization and search. However, we find several limitations in the original NFL paper. In this work, using results from the nature of search algorithms, we enhance several aspects of the original NFL Theorem. We have identified the properties of deterministic and probabilistic algorithms. We also provide an enumeration proof of the theorem. In addition, we show that the NFL Theorem is still valid for more general performance measures. This work serves as an application of the nature of search algorithms.**

Keywords*— *No Free Lunch, Nature of Search Algorithms, Optimization.

I. INTRODUCTION

Search techniques have been widely employed to solve different types of problems, ranging from sorting to optimization. The common objective of such search algorithms is to find an appropriate solution with desirable properties described by the problem. The general practice in solving search problems is: we have a problem and then try to propose an algorithm which can find the desirable solution of the problem among many other solutions. Most of the time, we keep on developing algorithms according to our experiences, intuition, and by trial-and-error. However, many problems in science and engineering can be reducible to the standard *NP*-hard or *NP*-complete problems [1]. According to the general belief in the research community, $P \neq NP$. In other words, attempting to find an algorithm which guarantees the desirable solution in polynomial time for these problems is futile.

Until the development of stochastic search algorithms a few decades ago, there were no easy and effective ways to handle *NP*-hard and *NP*-complete problems of large size. Stochastic search algorithms do not guarantee the best solutions in polynomial time, but they can normally generate good solutions for a wider range of problems in a reasonable period of time, when compared with deterministic algorithms. Suppose we compare the performance of a deterministic algorithm to that of a probabilistic algorithm. Assuming they have the same input and running time, the set of possible outputs generated by the latter must be larger. Each output may correspond to good performance for a particular problem. Thus, we conclude that the probability of a probabilistic algorithm generating outputs with good performance for a larger range of problems is higher. A "good" solution here means that it is within certain tolerance of the best solution. In science and engineering, good solutions are generally sufficient to fulfill the requirement of various objectives.

Most of the stochastic search algorithms are nature-inspired. Examples include Genetic Algorithm (GA) [2, 3], Simulated Annealing [4], Ant Colony Optimization (ACO) [5, 6], and Chemical Reaction Optimization [7]. There are also algorithms which are not adapted from natural processes, e.g., Tabu Search [8]. We generally classify the algorithms into heuristics and metaheuristics. The former refers to those which are specialized to address certain classes of problems, but normally give unfavorable solutions to other problems. The reason is that problem-specific information has been embedded in the algorithm development process, and thus, they are tailored to the specific problems very well. However, when they are applied to other problems, the embedded problem-specific information will not be useful anymore and they result in bad performance. On the other hand, the latter is not tailor-made for any particular problems and does not contain any problem-specific information. When compared with heuristics, metaheuristics can be applied to a wider range of problems with good performance. Thus, there is an accuracy-flexibility tradeoff between heuristics and metaheuristics.

Most of the stochastic search algorithms are metaheuristics. Whenever a metaheuristic appears to work well on a particular problem, greedy approaches or heuristic components may be added. This is tantamount to including more problem-specific information to the metaheuristic and making it more heuristic-like. According to the No-Free-Lunch Theorem [9, 10], the resulting metaheuristic is no longer able to solve other problems as well as before. In other words, we sacrifice flexibility for accuracy.

Different metaheuristics are developed based on distinct underlying mechanisms or phenomena. For example, GA is based on the idea of natural selection of living organisms while ACO makes use of the ecological behavior of ants in finding food. Although metaheuristics can be applied to a wide range of problems, they do have different performance when applied to different classes of problems. Then it is natural to ask "Is it possible to have a metaheuristic which is universally better than the others?" "Universally better" can be interpreted in the sense that the metaheuristic can solve more problems with better solutions. Some research is dedicated to developing universally better metaheuristics and it is claimed that some search algorithms can beat others on average [2].

Most research focuses on the construction of search algorithms. There was hardly any work on the theoretical

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

foundation until Wolpert and Macready proposed the No-Free-Lunch (NFL) Theorem [9, 10]. It states that all search algorithms perform statistically identically on solving computation problems. In other words, it is theoretically impossible to have a best general-purpose universal search strategy [11]. The NFL Theorem is controversial and raises many discussions (e.g. [12]) on its impact and correctness.

The search for a best general-purpose metaheuristic will be futile if the NFL Theorem holds. We believe that the theorem is correct and the conservation of performance for search exists. As the NFL results are important, we examine [9, 10] carefully again. We enhance and demonstrate the theorem with the nature of search algorithms [13]. The contributions of this work include clarifying the properties of NFL-governed algorithms, providing a verification method of the theorem, and generalizing the performance measures. The rest of this paper is organized as follows. Section II describes the terminology used in search and optimization, and the NFL Theorem and its extensions. In Section III, we introduce the nature of search algorithms. We demonstrate some enhancements and an enumeration proof of the theorem in Section IV and conclude this paper in Section V.

II. Related Work

A. Terminology

We describe a search problem as a function $f \in F$ with an argument $x \in X$ and a functional value $y \in Y$, i.e. $f(x) = y$. X and Y are discrete and finite[1]. We assume that their cardinalities are equal to $|X|$ and $|Y|$, respectively, and thus, $|F| = |Y|^{|X|}$. Let a be a deterministic non-potentially retracing search algorithm [9, 10]. At iteration $m \in \{1,2,...,|X|-1\}$, a takes a history $d_m = [(x_1, y_1),(x_2, y_2),...,(x_m, y_m)]$ as input and outputs x_{m+1}, i.e. $a(d_m) = x_{m+1}$. (We also use d to refer to a history in general.) Let d_m^x and d_m^y denote the x- and y-components of d_m. "Deterministic" means that a produces the same x_{m+1} with probability one whenever given the same d_m, while "non-potentially retracing" indicates $x_{m+1} \notin d_m^x$.

B. An Overview of the NFL Theorem

The NFL Theorem is firstly proposed in [9, 10], and consists of several results. The most representable one, which establishes the core idea, is *Theorem 1* in [10]. Mathematically, it states that for any pair of deterministic algorithms a_1 and a_2,

$$\sum_{f \in F} P(d_m^y \mid f, m, a_1) = \sum_{f \in F} P(d_m^y \mid f, m, a_2), \qquad (1)$$

where $P(d_m^y \mid f, m, a_1)$ represents the probability of having d_m^y given f, m and a_1. Note that each f is taken from F with the same probability. Equation (1) means that for any possible m, the performance of an algorithm (which can be obtained from $P(d_m^y)$) is independent of the algorithm itself, when every $f \in F$ is evaluated once. In other words, all algorithms (heuristics or metaheuristics) have equal performance when averaged over all possible functions. However, the theorem does not hold when we focus on a certain class of problems. Thus it is still possible for an algorithm to outperform another one in this situation.

C. Extensions

The original NFL Theorem requires all possible functions to be evaluated. Some researchers think that this requirement is far from reality as some functions are more practical (i.e. have higher probability of occurrence) than the others. In [14], the authors consider some special subsets of F and propose the sharpened NFL Theorem. It states that the original NFL Theorem stills hold in a subset of F provided that the subset is closed under permutation (c.u.p.). Igel and Toussaint study the necessary and sufficient conditions such that the NFL result holds in non-uniform distributions of functions in a c.u.p. subset [15]. Steeter investigates classes of functions where the NFL Theorem does not hold [16] while an Almost No Free Lunch Theorem is proposed to demonstrate how the result can hold approximately [17]. More people have realized the importance and impact of the NFL Theorem. The research community has been getting more interested in it and related papers are published every year.

III. Nature of Search Algorithms

Previously, we worked out the nature of search algorithms, which is focused on the algorithmic perspective, instead of the functional viewpoint [18]. We will highlight the major results below. For more details and proofs, interested readers can refer to [13].

We adopt the same working principle of search algorithms[2], described in Subsection II-A. With the sets X and Y defined, we can determine the whole set of functions F. Algorithms work on functions and generate histories. The analysis starts by enumerating all possible histories, which include all the results, i.e. (x, y) pairs, generated by any possible algorithms evaluating every $f \in F$ (the way to enumerate histories will be explained soon). We group the histories in a table and name it an enumeration table of histories. Table I(a) shows the enumeration table for the example with $X = \{1,2,3\}$ and $Y = \{1,2,3\}$ (you can ignore the shadings, boxes and the last column about counting the number of shaded entries in boxes at this moment). The construction of an enumeration table is easy. Each row and column correspond to a function and a permutation of X, respectively. For example, there are 27 ($= 3^3$) rows and 6 ($= 3!$) columns in Table I(a). A permutation of X is,

[1] The discreteness and finiteness are trivial if we employ a digital computer to solve the problems. X and Y represented in a computer must be discrete and finite.

[2] Every algorithm should be given an input, which is equal to the initial point where the search should start. All algorithms should not be restricted to fixed starting points. Otherwise, they are just pieces of codes to illustrate the pre-defined search paths, instead of tools which we can utilize to solve different problems.

TABLE I. ENUMERATION TABLES OF HISTORIES FOR X={1,2,3} AND Y={1,2,3}

(a)

Table Entries are *y*-components of the Histories		*x*-components of the Histories																	Number of Histories both in Shading and Box
		3	***2***	***1***	***3***	***1***	***2***	***2***	***3***	***1***	***2***	***1***	***3***	***1***	***2***	***3***	***1***	***3***	***2***	
Functions	f_1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	f_2	1	1	2	1	2	1	1	1	2	1	2	1	2	1	1	2	1	1	1
	f_3	1	1	3	1	3	1	1	1	3	1	3	1	3	1	1	3	1	1	0
	f_4	1	2	1	1	1	2	2	1	1	2	1	1	1	2	1	1	1	2	1
	f_5	1	2	2	1	2	2	2	1	2	2	2	1	2	2	1	2	1	2	0
	f_6	1	2	3	1	3	2	2	1	3	2	3	1	3	2	1	3	1	2	0
	f_7	1	3	1	1	1	3	3	1	1	3	1	1	1	3	1	1	1	3	0
	f_8	1	3	2	1	2	3	3	1	2	3	2	1	2	3	1	2	1	3	0
	f_9	1	3	3	1	3	3	3	1	3	3	3	1	3	3	1	3	1	3	0
	f_{10}	2	1	1	2	1	1	1	2	1	1	1	2	1	1	2	1	2	1	1
	f_{11}	2	1	2	2	2	1	1	2	2	1	2	2	2	1	2	2	2	1	1
	f_{12}	2	1	3	2	3	1	1	2	3	1	3	2	3	1	2	3	2	1	0
	f_{13}	2	2	1	2	1	2	2	2	1	2	1	2	1	2	2	1	2	2	2
	f_{14}	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	0
	f_{15}	2	2	3	2	3	2	2	2	3	2	3	2	3	2	2	3	2	2	0
	f_{16}	2	3	1	2	1	3	3	2	1	3	1	2	1	3	2	1	2	3	1
	f_{17}	2	3	2	2	2	3	3	2	2	3	2	2	2	3	2	2	2	3	0
	f_{18}	2	3	3	2	3	3	3	2	3	3	3	2	3	3	2	3	2	3	0
	f_{19}	3	1	1	3	1	1	1	3	1	1	1	3	1	1	3	1	3	1	0
	f_{20}	3	1	2	3	2	1	1	3	2	1	2	3	2	1	3	2	3	1	1
	f_{21}	3	1	3	3	3	1	1	3	3	1	3	3	3	1	3	3	3	1	0
	f_{22}	3	2	1	3	1	2	2	3	1	2	1	3	1	2	3	1	3	2	1
	f_{23}	3	2	2	3	2	2	2	3	2	2	2	3	2	2	3	2	3	2	0
	f_{24}	3	2	3	3	3	2	2	3	3	2	3	3	3	2	3	3	3	2	0
	f_{25}	3	3	1	3	1	3	3	3	1	3	1	3	1	3	3	1	3	3	0
	f_{26}	3	3	2	3	2	3	3	3	2	3	2	3	2	3	3	2	3	3	0
	f_{27}	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	0

(b)

Table Entries are *y*-components of the Histories		*x*-components of the Histories																	Number of Histories both in Shading and Box
		3	***2***	***1***	***3***	***1***	***2***	***2***	***3***	***1***	***2***	***1***	***3***	***1***	***2***	***3***	***1***	***3***	***2***	
Functions	f_1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	f_2	1	1	2	1	2	1	1	1	2	1	2	1	2	1	1	2	1	1	1
	f_3	1	1	3	1	3	1	1	1	3	1	3	1	3	1	1	3	1	1	0
	f_4	1	2	1	1	1	2	2	1	1	2	1	1	1	2	1	1	1	2	1
	f_5	1	2	2	1	2	2	2	1	2	2	2	1	2	2	1	2	1	2	1
	f_6	1	2	3	1	3	2	2	1	3	2	3	1	3	2	1	3	1	2	0
	f_7	1	3	1	1	1	3	3	1	1	3	1	1	1	3	1	1	1	3	0
	f_8	1	3	2	1	2	3	3	1	2	3	2	1	2	3	1	2	1	3	1
	f_9	1	3	3	1	3	3	3	1	3	3	3	1	3	3	1	3	1	3	0
	f_{10}	2	1	1	2	1	1	1	2	1	1	1	2	1	1	2	1	2	1	1
	f_{11}	2	1	2	2	2	1	1	2	2	1	2	2	2	1	2	2	2	1	1
	f_{12}	2	1	3	2	3	1	1	2	3	1	3	2	3	1	2	3	2	1	1
	f_{13}	2	2	1	2	1	2	2	2	1	2	1	2	1	2	2	1	2	2	1
	f_{14}	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	0
	f_{15}	2	2	3	2	3	2	2	2	3	2	3	2	3	2	2	3	2	2	0
	f_{16}	2	3	1	2	1	3	3	2	1	3	1	2	1	3	2	1	2	3	0
	f_{17}	2	3	2	2	2	3	3	2	2	3	2	2	2	3	2	2	2	3	0
	f_{18}	2	3	3	2	3	3	3	2	3	3	3	2	3	3	2	3	2	3	0
	f_{19}	3	1	1	3	1	1	1	3	1	1	1	3	1	1	3	1	3	1	0
	f_{20}	3	1	2	3	2	1	1	3	2	1	2	3	2	1	3	2	3	1	0
	f_{21}	3	1	3	3	3	1	1	3	3	1	3	3	3	1	3	3	3	1	0
	f_{22}	3	2	1	3	1	2	2	3	1	2	1	3	1	2	3	1	3	2	1
	f_{23}	3	2	2	3	2	2	2	3	2	2	2	3	2	2	3	2	3	2	0
	f_{24}	3	2	3	3	3	2	2	3	3	2	3	3	3	2	3	3	3	2	0
	f_{25}	3	3	1	3	1	3	3	3	1	3	1	3	1	3	3	1	3	3	0
	f_{26}	3	3	2	3	2	3	3	3	2	3	2	3	2	3	3	2	3	3	0
	f_{27}	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	0

in fact, one possible $d^x_{|X|}$. An entry gives the $d^y_{|X|}$ of the corresponding $d^x_{|X|}$ and *f*. Note that an enumeration table is a complete description of search problems. Although its size grows exponentially with $|X|$ and $|Y|$, the entries can be easily generated by inputting all permutations of *X* into the functions. Important properties of search algorithms can be determined from the enumeration table.

We consider deterministic algorithms first and obtain the following results:

Lemma 1. Consider a deterministic algorithm *a* which is a mapping from the set containing all possible *d* to the set containing all possible *x*, i.e. $a(d) = x$. If there are *m* and *n* possibilities for *d* and *x*, respectively, there will be n^m possible different *a*.

Theorem 1. Given a system with the sets *X* and *Y*, with $|X| \geq 3$ and $|Y| \geq 2$, there are $\prod_{i=2}^{|X|-1} i^{|X||Y|^{|X|-i}}$ possible different deterministic algorithms.

Corollary 1. Given a system with the sets *X* and *Y*, with $|X| \geq 3$ and $|Y| \geq 2$, the set of all deterministic algorithms is finite. We can list all of them by indicating the choices of x_2, x_3, ..., and $x_{|X|-1}$ one-by-one for all the partial histories $d_1, d_2, \ldots$, and $d_{|X|-2}$, respectively.

Theorem 2. For $1 \leq i \leq |X|$, let $d^y_{|X|}(i)$ be the value of y_i specified in $d^y_{|X|}$. Given a system with sets *X* and *Y*, with $|X| \geq 3$ and $|Y| \geq 2$, any specific $[d^y_{|X|}(1), d^y_{|X|}(2), \ldots, d^y_{|X|}(i)]$ can be determined the same number of times in the enumeration table by each possible deterministic algorithm.

We find that every algorithm can be represented by a logic matrix [13]. Table II gives two examples of such matrices deduced from the enumeration table shown in Table I(a). A logic matrix can be interpreted like this: the 1st column stands for x_1, the 2nd for y_1, the 3rd for x_2, and so on. A row represents a history generated by that algorithm. We also call a row a primitive logic because it gives an if-then relationship between the past history d_m and the next point x_{m+1} for *m* = 1, 2,…, |*X*|-1. For example, the first primitive logic of Algorithm 1 can be interpreted as: if $[(x_1, y_1)] = [(1,1)]$, then $x_2 = 2$. The whole matrix, in fact, characterizes all possible histories which can be generated by that algorithm.

The analysis can be extended to probabilistic search algorithms [13]. A probabilistic algorithm is defined similarly as a deterministic algorithm, but instead of mapping to a single x_{m+1}, it maps to multiple x_{m+1} with different probabilities from d_m. By considering the logic matrices and their primitive logics, we obtain the following results:

Theorem 3. Given a system with the sets *X* and *Y*, with $|X| \geq 3$ and $|Y| \geq 2$, the corresponding deterministic algorithms form the basis of its algorithmic space, which consists of all deterministic and probabilistic algorithms. All algorithms are formed by probabilistically combining some of the deterministic algorithms.

TABLE II. LOGIC MATRICES OF TWO DETERMINISTIC ALGORITHMS

Algorithm 1	**Algorithm 2**
1 1 2	1 1 2
1 2 2	1 2 3
1 3 2	1 3 2
2 1 1	2 1 3
2 2 1	2 2 1
2 3 1	2 3 3
3 1 1	3 1 1
3 2 1	3 2 2
3 3 1	3 3 1

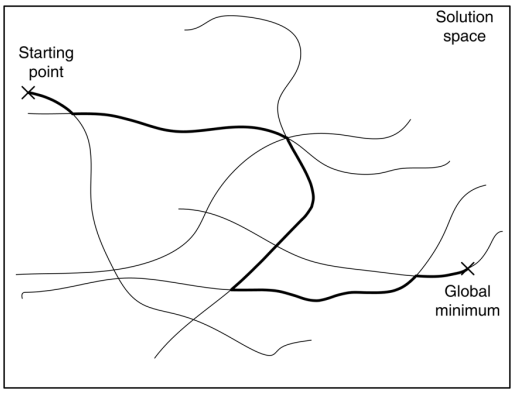

Figure 1. An illustration of a probabilistic algorithm with a set of deterministic algorithms.

Corollary 2. Given a system with the sets *X* and *Y*, with $|X| \geq 3$ and $|Y| \geq 2$, an algorithm, both deterministic and probabilistic, can be represented by a probabilistic distribution on the set of the deterministic algorithms.

We can demonstrate the above results with the example shown in Fig. 1, which depicts a scenario of solving a minimization problem[3]. In the figure, the rectangle represents the solution space, which is composed of all feasible solutions, *x*, of the problem. Each thin line gives a deterministic algorithm [4] in which the order of picking *x* has been deterministically defined when the algorithm is built. The thick line describes the resulting search path of a probabilistic algorithm. Along the path from the starting point to the global minimum, there are intersection points where the algorithm has more than one alternative path to choose from. This exactly fulfills the definition of probabilistic algorithms. In fact, *Theorem 3* and *Corollary 2* tell us that a probabilistic algorithm is characterized by a combination of deterministic algorithms,

[3] The scenario shown in Fig. 1 is not typical. The function is ignored. The actual cases should be much more complicated. For simplicity, and without loss of generality, we use this simplified scenario to illustrate the results.

[4] Here we characterize a deterministic algorithm with a partial *x*-component of one of its possible computable histories. In reality, it should be represented by a set of histories which traverse all solution points and each history has a different starting point.

i.e. the thin lines. The behavior of the probabilistic algorithm is intrinsically defined by the combination of the underlying deterministic algorithms. If we know which and how the underlying deterministic algorithms are combined, we can infer the performance of the probabilistic algorithm.

We will make use of the above results to re-investigate several issues related to the NFL Theorem.

IV. A Re-Visit of the NFL Theorem

We study the details of the NFL Theorem again, mainly from [10]. There are several possible enhancements of the theorem. We discuss the aspects of optimization algorithms, verification of the theorem, and performance measures accordingly in the following subsections.

A. Optimization Algorithms

In [10], a deterministic optimization algorithm is defined as a mapping from previously visited sets of points to a single previously unvisited point in X, i.e. $a: d_m \in D_m \mapsto \{x \mid x \notin d_m^x\}$. Then the NFL result (e.g. (1)) is developed based on a_1, a_2,…, which are different representations of the general term a. This implies all a_1, a_2,… form the set of deterministic algorithm, A_d, i.e. a_1, a_2,… $\in A_d$ [5]. But what does A_d indeed look like? Is the set finite? From *Theorem 1* and *Corollary 1* in Section III, we have already given the answers. A_d is finite and all the elements can be listed in the form of logic matrices.

Then the authors of [10] defines a probabilistic (i.e. stochastic) optimization algorithm, σ, as a mapping from $d_m \in D_m$ to a d_m -dependent distribution over X with zero probability for all $x \in d_m^x$. Let A_p be the set of all possible probabilistic algorithms. The NFL result for deterministic algorithms is also valid for probabilistic algorithms by replacing a with σ throughout the proof, as explained in [10]. Mathematically, (1) becomes

$$\sum_{f \in F} P(d_m^y \mid f, m, \sigma_1) = \sum_{f \in F} P(d_m^y \mid f, m, \sigma_2) \tag{2}$$

for any pair of algorithms σ_1 and σ_2 in A_p. In this way, the NFL result is legitimate for any algorithms in A_d and A_p, respectively. The immediate follow-up question is whether the NFL result is still true when we consider any algorithms in both A_d and A_p at the same time, i.e., for any $\alpha_1, \alpha_2 \in A_d \cup A_p$,

$$\sum_{f \in F} P(d_m^y \mid f, m, \alpha_1) = \sum_{f \in F} P(d_m^y \mid f, m, \alpha_2). \tag{3}$$

We will prove (3) in Subsection C.

B. Verification of the NFL Theorem

As we can determine the important properties of search from the enumeration table and the NFL Theorem is one of the fundamental results of search, it is logical to deduce the NFL Theorem from the enumeration table. In the following, we try to look at the NFL result, i.e. (1), in the implementation perspective, instead of the original approach based on probability theory [10].

For any system with X and Y specified, we can produce the corresponding enumeration table (as explained in Section III). We now consider deterministic algorithms first. By *Theorem 1* and *Corollary 1* in Section III, we can list all $a \in A_d$ in the form of logic matrices. We can check the NFL result with the following procedures:

1. Select a specific d_m^y for $1 \le m \le |X|$;
2. For each entry in the enumeration table, shade it if it matches d_m^y;
3. For each $a \in A_d$,
   i. Put all possible y-components of the histories which can be generated by a in boxes;
   ii. Count how many entries are both in shadings and in boxes for each function, and calculate the total for all functions;
   iii. Delete all the boxes;
4. *Theorem 2* in Section III guarantees all the totals (of all $a \in A_d$) are the same.

The shaded entries and those in boxes represent those y-components of histories which match d_m^y, and those which can be generated by a, respectively. The shaded ones in boxes denote computable histories with good performance by a. The total number of shaded entries of a in boxes reflects its "ability" to generate good results. In Step 3(ii), the direct summation to get the total implies that we give equal weight to every function. After finding the total, we have finished the examination of a particular a. Step 3(iii) restores the setting for d_m^y only and makes it ready for another a. The positive answer at Step 4 tells us that the NFL result, i.e., all deterministic search algorithms, $a \in A_d$, perform statistically identically on solving computation problems $f \in F$.

Note that the performance of an algorithm is based on the histories obtained during execution and the histories are generated by the algorithm itself. Thus, the NFL results should be understood from the algorithmic perspective. On the other hand, the above method originates from the enumeration table, whose generation is independent of the execution of the algorithms. Therefore, we approach the NFL Theorem from a different angle. In fact, the above method provides an enumeration proof (also called exhaustive proof) for the NFL Theorem. Enumeration proof [6] [20] is valid when the number of test cases is finite. From *Theorem 1* in Section III, the set of deterministic algorithms is finite, and thus, enumeration proof is valid.

[5] Although A_d is not explicitly indicated in [10], it is trivial to see that all deterministic algorithms will form the set A_d.

[6] An example of using enumeration proof to validate mathematical statements can be found in [19].

To demonstrate the above method, we consider the system with $X = \{1,2,3\}$ and $Y = \{1,2,3\}$ again. Let $m = 2$. We define d_2^y equal to [2 1]. Table I shows the results of completing up to Step 3(ii) for the two deterministic algorithms shown in Table II (Table I(a) and I(b) correspond to Algorithms 1 and 2, respectively). It is easy to check both the totals of Algorithm 1 and 2 are equal to 9 by summing up the last columns of Table I(a) and I(b).

C. Performance Measures

In [10], performance measures are indicated by $\Phi(d_m^y)$, which can be interpreted as any function with the *y*-component of the history as the input. The performance is based on d_m^y only. For example, $\Phi(d_m^y) = \min_i\{d_m^y(i) : i = 1,...,m\}$ for minimizing f [7].

For optimization, we doubt the performance measure should be based solely on d_m^y. When we are optimizing *f* with an algorithm *a*, we want to evaluate the performance of *a* with respect to the most desirable function values of *f*. Without considering *f*, the actual values of d_m^y computed by *a* are meaningless. We can illustrate the above idea with the following example:

Suppose we try to minimize two different functions f_1 and f_2 (see Fig. 2) with the same optimization algorithm *a*. We are only interested in evaluating the performance of *a* after choosing the first three *x* in *X*, i.e. *m* = 3. If *a* generates the same $d_3^y = [4,3,2]$ for both f_1 and f_2, can we say that *a* has better performance on either f_1 or f_2 by solely looking at d_3^y? If we have some ideas about all the *y*-values of the functions (e.g. we have the curves of the functions as in Fig. 2), we can definitely conclude that *a* has better performance on f_1 than on f_2 because the minimum of f_1 has been obtained in d_3^y, but not for f_2. By only looking at d_3^y, however, *a* has the same performance on both functions. Thus, the performance measure should be function-dependent, and it should reflect the relative function values in d_m^y to the most desirable function values in question (e.g., for minimizing *f* with minimum *y**, the performance measure can check the ratio of $y \in d_m^y$ to *y**). More general performance measures can then be indicated by $\Phi_f(d_m^y)$. Let $D_m^y(\phi) = \{d_m^y : \Phi_f(d_m^y) \text{ satisfy } \phi\}$, where ϕ can be any reasonable conditions for performance evaluation. For example, ϕ can be specified to require $y_2 = \min\{y \in Y\}$ when *m* = 3.

Equation (1), i.e. the key formula which constitutes the NFL Theorem, also inherits the above mentioned problem. The core of (1) is $P(d_m^y \mid f,m,a)$. In [10], it is interpreted as "the conditional probability of obtaining a particular sample d_m under the condition that an algorithm *a* iterates *m* times on a cost function *f*". Then the performance of *a* is said to be measured with $P(d_m^y \mid f,m,a)$, by stating that $P(\Phi(d_m^y) \mid f,m,a)$ can be easily derived from $P(d_m^y \mid f,m,a)$.

[7] This is the original example explaining performance measures in [10].

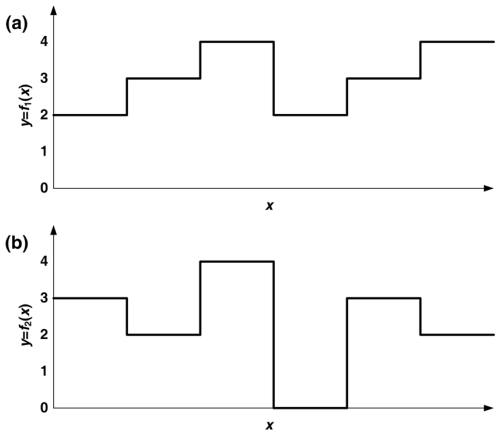

Figure 2. Examples to illustrate the effect of the performance measure.

If we adopt a more general $\Phi_f(d_m^y)$ as the performance measure, the NFL Theorem should be re-stated as: the average over all f of $P(\Phi_f(d_m^y) \mid f,m,a)$ is independent of *a*, with $\Phi(d_m^y)$ being replaced by $\Phi_f(d_m^y)$. This result can in turn be derived by showing that for any pair of deterministic algorithms a_1 and a_2,

$$\sum_{f \in F} P(D_m^y(\phi) \mid f,m,a_1) = \sum_{f \in F} P(D_m^y(\phi) \mid f,m,a_2) . \qquad (4)$$

We are not trying to say that (1) fails to come up with the more reasonable interpretation of performance measures, but we can show that (1) can be used to derive (4). We can obtain the connection by observing the enumeration table of any system with *X* and *Y* specified. We try to demonstrate the connection with the simple example shown in Table I(a). We first discuss d_m^y in (1). The system has *X* and *Y* with |*X*| = |*Y*| = 3. Thus there are 6 (= |*X*|! = 3!) columns of d_3^y and each entry in a column is the d_3^y of the corresponding *f*. As |*Y*| = 3, the maximum length of d_3^y is three. For any particular d_m^y, there are exactly $|Y|^{|X|}/|Y|^m$ corresponding entries in each column. Their locations may be different in different columns, i.e. the same d_m^y appears at rows corresponding to different functions in different columns. For example, $d_1^y = [1]$, $d_2^y = [1,2]$ and $d_3^y = [1,2,3]$ has exactly 9 ($=|Y|^{|X|}/|Y|$), 3 ($=|Y|^{|X|}/|Y|^2$) and 1 ($=|Y|^{|X|}/|Y|^3$) entries in each column, respectively.

We now come to $D_m^y(\phi)$ in (4). Recall that $D_m^y(\phi)$ is a set of d_m^y with the condition ϕ satisfied. When checking $D_m^y(\phi)$ from column to column in an enumeration table, we find that $D_m^y(\phi)$ appears exactly once in every column but $d_m^y \in D_m^y(\phi)$ may correspond to different *f*. In other words, we can consider each d_m^y in $D_m^y(\phi)$ one-by-one and apply (1) to each of them. We have

$$\sum_{d_m^y \in D_m^y(\phi)} \sum_{f \in F} P(d_m^y \mid f, m, a_1) = \sum_{d_m^y \in D_m^y(\phi)} \sum_{f \in F} P(d_m^y \mid f, m, a_2) ,$$

which can further yield (4). From this, we are confident that the NFL Theorem holds for any reasonable performance measure.

Up to now, we have verified (4) (and (1) which is a special case of (4)). We are going to validate (3). Consider probabilistic algorithms, $\sigma \in A_p$. By *Theorem 3* in Section III, all probabilistic algorithms can be expressed as a probability distribution over the set of defined deterministic algorithms. Let the sample space Ω equal A_d. We can model each probabilistic algorithm σ by a discrete random variable H with probability mass function $p(h)$, where h is any possible outcome in Ω, i.e., $\sigma = \sum_{h \in \Omega} hp(h), \sigma \in A_p$. Thus, for any $a \in A_d$ and $\sigma \in A_p$,

$$\begin{aligned} \sum_{f \in F} P(d_m^y \mid f, m, \sigma) &= \sum_{h \in \Omega} \sum_{f \in F} P(d_m^y \mid f, m, h) p(h) \\ &= \sum_{f \in F} P(d_m^y \mid f, m, h) \sum_{h \in \Omega} p(h) \\ &= \sum_{f \in F} P(d_m^y \mid f, m, d) . \end{aligned} \quad (5)$$

This means that the performance of any deterministic algorithm and that of any probabilistic algorithm are the same when averaged over all *f*. By combining (1), (2) and (5), we get (3). Similar to the way of getting (4) from (1), we have an immediate corollary that for any $\alpha_1, \alpha_2 \in A_d \cup A_p$,

$$\sum_{f \in F} P(D_m^y(\phi) \mid f, m, \alpha_1) = \sum_{f \in F} P(D_m^y(\phi) \mid f, m, \alpha_2) .$$

We know that deterministic and probabilistic algorithms constitute all optimization algorithms. Therefore, any reasonable performance measure of any algorithms (deterministic and/or probabilistic) is statistically identical when averaged over all *f*.

V. Conclusion

Before the NFL Theorem was published, most research on optimization focused on solving specific problems with proposed algorithms. The theoretical foundation was largely ignored. The NFL Theorem draws our attention to the theoretical foundation and demonstrates a fundamental limit of all search algorithms. However, we find limitations of NFL statements in the original NFL papers. By using results from the nature of search algorithms, we enhance several aspects of the original NFL Theorem. We have identified the properties of deterministic and probabilistic algorithms. We provide an enumeration proof. In addition, we generalize the NFL results with more general performance measures and remove the limitations.

References

[1] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. New York, NY: W. H. Freeman & Co Ltd, 1979.

[2] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning. Reading, MA: Addison-Wesley, 1989.

[3] J.H. Holland, Adaptation in Natural and Artificial Systems. Cambridge, MA: MIT Press, 1992.

[4] S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi, "Optimization by Simulated Annealing," Science, vol. 220, no. 4598, pp. 671-680, May 1983.

[5] E. Bonabeau, M. Dorigo, and G. Theraulaz, "Inspiration for Optimization from Social Insect Behaviour," Nature, vol. 406, no. 6791, pp. 39-42, July 2000.

[6] M. Dorigo and T. Stützle, Ant Colony Optimization. Cambridge, MA: MIT Press, 2004.

[7] A.Y.S. Lam and V.O.K. Li, "Chemical-Reaction-Inspired Metaheuristic for Optimization," submitted for publication.

[8] F. Glover and F. Laguna, Tabu Search. Norwell, MA: Kluwer Academic Publishers, 1997.

[9] D.H. Wolpert and W.G. Macready, "No Free Lunch Theorems for Search," Santa Fe Institute, Santa Fe, NM, Tech. Rep. 95-02-010, 1995.

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

[11] Y.C. Ho and D.L. Pepyne, "Simple Explanation of the No-Free-Lunch Theorem and Its Implications," Journal of Optimization Theory and Applications, vol. 115, no. 3, pp. 549-570, December 2002.

[12] J.C. Culberson, "On the futility of blind search: An algorithmic view of "No Free Lunch," Evolutionary Computation, vol. 6, no. 2, pp. 109-127, 1998.

[13] A.Y.S. Lam and V.O.K. Li, "Nature of Search Algorithms," Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong, Tech. Rep. TR-2009-001, January 2009.

[14] C. Schumacher, M.D. Vose, and L.D. Whitley, "The No Free Lunch and Problem Description Length," In Proceedings of the Genetic and Evolutionary Computation Conference. San Francisco, CA, Morgan Kaufmann, pp. 565-570, 2001.

[15] C. Igel and M. Toussaint, "No-Free-Lunch theorem for non-uniform distributions of target functions," Journal of Mathematical Modelling and Algorithms, vol. 3 no. 4, pp. 313-322, 2004.

[16] M.J. Streeter, "Two broad classes of functions for which a no free lunch result does not hold," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pp. 1418-1430, 2003.

[17] S. Droste, T. Jansen, and I. Wegener, "Optimization with randomized search heuristics - The (A)NFL theorem, realistic scenarios, and difficult functions," Theoretical Computer Science, vol. 287, no. 1, pp. 131-144, September 2002.

[18] T. English, "No more lunch: Analysis of sequential search," in Proceedings of the 2004 Congress on Evolutionary Computation, Portland, OR, pp. 227-234, 2004.

[19] J. Sloan, "Two Rectangles are Constructible with Tangrams: An Enumeartion Proof using Mathematica," Mathematica in Education, vol. 2, no. 4, pp. 7-10, 1993.

[20] J. Sloan, Discrete Mathematics. Upper Saddle River, NJ: Prentice Hall, 2008.