Abstract—Casting machine learning as a type of search, we demonstrate that the proportion of problems that are favorable for a fixed algorithm is strictly bounded, such that no single algorithm can perform well over a large fraction of them. If an algorithm greatly excels on a class of problems (e.g., convex problems), that class must necessarily be small. We give an upper bound on the expected performance for a search algorithm as a function of the mutual information between the target and the information resource (e.g., training dataset), proving the importance of certain types of dependence for machine learning. Lastly, given that the expected per-query probability of success for an algorithm is mathematically equivalent to a single-query probability of success under a distribution (called a search strategy), we prove that the proportion of favorable strategies is also strictly bounded. Thus, whether one holds fixed the search algorithm and considers all possible problems or one fixes the search problem and looks at all possible search strategies, favorable matches are exceedingly rare. The forte (strength) of any algorithm is quantifiably restricted.

I. INTRODUCTION

BIGCITY, United States. In a fictional world not unlike our own, a sophisticated criminal organization plots to attack an unspecified landmark within the city. Due to the complexity of the attack and methods of infiltration, the group is forced to construct a plan relying on the coordinated actions of several interdependent agents, of which the failure of any one would cause the collapse of the entire plot. As a member of the city’s security team, you must allocate finite resources to protect the many important locations within the city. Although you know the attack is imminent, your sources have not indicated which location, of the hundreds possible, will be hit; your lack of manpower forces you to make assumptions about target likelihoods. You know you can foil the plot if you stop even one enemy agent. Because of this, you seek to maximize the odds of capturing an agent by placing vigilant security forces at the strategic locations throughout the city. Allocating more security to a given location increases surveillance there, raising the probability a conspirator will be found if operating nearby. Unsure of your decisions, you allocate based on your best information, but continue to second-guess yourself.

With this picture in mind, we can analyze the scenario through the lens of algorithmic search. Algorithmic search methods, whether employed in fictional security situations or by researchers in the lab, all share common elements. They have a search space of elements (possible locations) which contains items of high value at unknown locations within that space. These high-value elements are targets of the search, which are sought by a process of inspecting elements of the search space to see if they contain any elements of the target set. (We refer to the inspection of search space elements as sampling from the search space, and each sampling action as a query.) In our fictional scenario, the enemy activity locations constitute the target set, and surveillance at city locations corresponds to a search process (i.e., attempting to locate and arrest enemy agents). The search process can be deterministic or contain some degree of randomization, such as choosing which elements to inspect using a weighted distribution over the search space. The search process is a success if an element of the target set is located during the course of inspecting elements of the space. The history of the search refers to the collection of elements already sampled and any accompanying information (e.g., the intelligence data gathered thus far by the security forces).

There are many ways to search such a space. Research on algorithmic search is replete with proposed search methods that attempt to increase the expected likelihood of search success, across many different problem domains. However, a direct result of the No Free Lunch theorems are that no search method is universally optimal across all search problems [1], [2], [3], [4], [5], [6], [7], [8], [9], since all algorithmic search methods have performance equivalent to uniform random sampling on the search space when uniformly averaged across any closed-under-permutation set of problem functions [4]. Thus, search (and learning) algorithms must trade weaker performance on some problems for improved performance on others [1], [10]. Given the fact that there exists no universally superior search method, we typically seek to develop methods that perform well over a large range of important problems. An important question naturally arises:

Is there a limit to the number of problems for which a search algorithm can perform well?

For search problems in general, the answer is yes, as the proportion of search problems for which an algorithm outperforms uniform random sampling is strictly bounded in relation to the degree of performance improvement. Previous results have relied on the fact that most search problems have many target elements, and thus are not difficult for random sampling [11], [12]. Since the typical search problem is not difficult for random sampling, it becomes hard to find a problem for which an algorithm can significantly and reliably outperform random sampling. While theoretically interesting, a more relevant situation occurs when we consider difficult search problems, which are those having relatively small target sets. We denote such problems as target-sparse problems or following [11], target-sparse functions. Note, this use of sparseness differs from that typical in the machine learning literature, since it refers to sparseness of the target space, not sparseness in the feature space. One focus of this paper is to prove results that hold even for target-sparse functions. As mentioned, Montañez [11] and English [12] have shown that uniform sampling does well on the majority of functions since they are not target-sparse, thus making problems for which an algorithm greatly outperforms random chance rare. If we restrict ourselves to only target-sparse functions, perhaps it would become easier to find relatively favorable problems since we would already know uniform sampling does poorly. The bar would already be low, so we would...
have to do less to surpass it, perhaps leading to a greater proportion of favorable problems within that set. We show why this intuition is incorrect, along with proving several other key results.

II. Contributions

First, we demonstrate that favorable search problems must necessarily be rare. Our work departs from No Free Lunch results (namely, that the mean performance across sets of problems is fixed for all algorithms) to show that the proportion of favorable problems is strictly bounded in relation to the inherent problem difficulty and the degree of improvement sought (i.e., not just the mean performance is bounded). Our results continue to hold for sets of objective functions that are not closed-under-permutation, in contrast to traditional No Free Lunch theorems. Furthermore, the negative results presented here do not depend on any distributional assumptions on the space of possible problems, such that the proportion of favorable problems is small regardless of which distribution holds over them in the real world. This directly answers critiques aimed at No Free Lunch results arguing against a uniform distribution on problems in the real world (cf. [13]), since given any distribution over possible problems, there are still relatively few favorable problems within the set one is taking the distribution over. For example, if an algorithm greatly excels on a class of problems such as convex problems, that class must necessarily be small, regardless of whether convex problems are more likely (or less likely) to actually be encountered in the real world. This crucial distinction cannot be stressed too strongly.

As a corollary, we prove the information costs of finding any favorable search problem is bounded below by the number of bits “advantage” gained by the algorithm on such problems. We do this by using an active information transform to measure performance improvement in bits [14], proving a conservation of information result [6], [14], [11] that shows the amount of information required to locate a search problem giving b bits of expected information is at least b bits. Thus, to get an expected net gain of information, the true distribution over search problems must be biased towards favorable problems for a given algorithm. This places a floor on the minimal information costs for finding favorable problems, somewhat reminiscent of the entertainingly satirical work on “data set selection” [15].

Another major contribution of this paper is to bound the single-query probability of success for an algorithm based on information resources. Namely, we relate the degree of dependence (measured in mutual information) between target sets and external information resources, such as objective functions, noisy measurements or sets of training data, to the maximum improvement in search performance. We prove that for a fixed target-sparse and given an algorithm with induced single-query probability of success \( q \),

\[
q \leq \frac{I(T; F) + D(P_T \| U_T) + 1}{I_0}
\]

where \( I(T; F) \) is the mutual information between target set \( T \) (as a random variable) and external information resource \( F \), \( D(P_T \| U_T) \) is the Kullback-Leibler divergence between the marginal distribution on \( T \) and the uniform distribution on target sets, and \( I_0 \) is the baseline information cost for the search problem due to sparseness. This simple equation takes into account degree of dependence, target sparseness, target function uncertainty, and the contribution of random luck. It is surprising that such well-known quantities appear in the course of simply trying to upper-bound the probability of success.

\[1\text{Noisy in the sense of inaccurate or biased, not in the sense of indeterministic.}\]

In the Appendix [16], we establish the equivalence between the expected per-query probability of success for an algorithm and the probability of a successful single-query search under some distribution, which we call a strategy. Each algorithm maps to a strategy, and we prove an upper-bound on the proportion of favorable strategies for a fixed problem. Thus, matching a search problem to a fixed algorithm or a search algorithm to a fixed problem are both provably difficult tasks, and the set of favorable items remains vanishingly small in both cases.

Lastly, we apply the results to a few examples.

III. Search Framework

We begin by formalizing our problem setting, search method abstraction and other necessary concepts.

A. The Search Problem

We abstract the search problem as follows. The search space, denoted \( \Omega \), contains the elements to be examined. We limit ourselves to finite, discrete search spaces, which entails no loss of generality when considering search spaces fully representable on physical computer hardware within a finite time. Let the target set \( T \subset \Omega \) be a nonempty subset of the search space. The set \( T \) can be represented using a binary vector the size of \( \Omega \), where an indexed position evaluates to 1 whenever the corresponding element is in \( T \) and 0 otherwise. Thus, each \( T \) corresponds to exactly one binary vector of length \( |\Omega| \), and vice versa. We refer to this one-to-one mapped binary vector as a target function and use the terms target function and target set interchangeably, depending on context. These target sets/functions will help us define our space of possible search problems, as we will see shortly.

Figure 1 shows two example target sets, in binary vector and generic target set form. The set on the left has strong (potentially exploitable) structure governing the placement of target elements, whereas the example on the right is more or less random. Thus, knowing the location of some target elements may or may not be able to help one find additional elements, and in general there may be any degree of correlation between the location of target elements already uncovered and those yet to be discovered.

Typically, elements from the search space \( \Omega \) are evaluated according to some external information resource, such as an objective function \( f \). We abstract this resource as simply a finite length bit string, which could represent an objective function, a set of training data (in supervised learning settings), or anything else. The resource can exist in coded form, and we make no assumption about the shape of this resource or its encoding. Our only requirement is that it can
be used as an oracle, given an element (possibly the null element) from \( \Omega \). In other words, we require the existence of two methods, one for initialization (given a null element) and one for evaluating queried points in \( \Omega \). We assume both methods are fully specified by the problem domain and external information resource. With a slight abuse of notation, we define an information resource evaluation as \( F(\omega) := q(F, \omega) \), where \( q \) is an extraction function applied to the information resource and \( \omega \in \Omega \cup \{\emptyset\} \). Therefore, \( F(\emptyset) \) represents the method used to extract initial information for the search algorithm (absent of any query), and \( F(\omega) \) represents the evaluation of point \( \omega \) under resource \( F \). The size of the information resource becomes important for datasets in machine learning, which determines the maximum amount of mutual information available between \( T \) and \( F \).

A search problem is defined as a 3-tuple, \((\Omega, T, F)\), consisting of a search space, a target subset of the space and an external information resource \( F \), respectively. Since the target locations are hidden, any information gained by the search concerning the target locations is mediated through the external information resource \( F \) alone. Thus, the space of possible search problems includes many deceptive search problems, where the external resource provides misleading information about target locations, similar to when security forces are given false intelligence data, and many noisy problems, similar to when imprecise intelligence gathering techniques are used. In the fully general case, there can be any relationship between \( T \) and \( F \). Because we consider any and all degrees of dependence between external information resources and target locations, this effectively creates independence when considering the set as a whole, allowing the first main result of this paper to follow as a consequence.

However, in many natural settings, target locations and external information resources are tightly coupled. For example, we typically threshold objective function values to designate the target elements as those that meet or exceed some minimum. Doing so enforces dependence between target locations and objective functions, where the former is fully determined by the latter, once the threshold is known. This dependence causes direct correlation between the objective function (which is observable) and the target locations (which are not directly observable). We will demonstrate such correlation is exploitable, affecting the upper bound on the expected probability of success.

B. The Search Algorithm

An algorithmic search is any process that chooses elements of a search space to examine, given some history of elements already examined and information resource evaluations. The history consists of two parts: a query trace and a resource evaluation trace. A query trace is a record of points queried, indexed by time. A resource evaluation trace is a record of partial information extracted from \( F \), also indexed by time. The information resource evaluations can be used to build elaborate predictive models (as is the case of Bayesian optimization methods), or ignored completely (as is done with uniform random sampling). The algorithm’s internal state is updated at each time step (according to the rules of the algorithm), as new query points are evaluated against the external information resource. The search is thus an iterative process, with the history \( h \) at time \( i \) represented as \( h_i = (\omega_i, F(\omega_i)) \). The problem domain defines how much initial information from \( F \) is given by \( F(\emptyset) \), as well as how much (and what) information is extracted from \( F \) at each query.

We allow for both deterministic and randomized algorithms, since deterministic algorithms are equivalent to randomized algorithms with degenerate probability functions (i.e., they place all probability mass on a single point). Furthermore, any population based method can also be represented in this framework, by holding \( P \) fixed when selecting the \( m \) elements in the population, then considering the history (possibly limiting the horizon of the history to only \( m \) steps, creating a Markovian dependence on the previous population) and updating the probability distribution over the search space for the next \( m \) steps.

Abstraction away the details of how such a distribution \( P \) is chosen, we can treat the search algorithm as a black box that somehow chooses elements of a search space to evaluate. A search is successful if an element in the target set is located during the search. Algorithm 1 outlines the steps followed by the black-box search algorithm and Figure 2 visually demonstrates the process.

![Figure 2. Black-box search algorithm. At time \( i \) the algorithm computes a probability distribution \( P_i \) over the search space \( \Omega \), using information from the history, and a new point is drawn according to \( P_i \). The point is evaluated using external information resource \( F \). The tuple \((\omega, F(\omega))\) is then added to the history at position \( i \). Note, indices on \( \omega \) elements do not correspond to time step in this diagram, but to sampled locations.](image)

### Algorithm 1 Black-box Search Algorithm

1. Initialize \( h_0 \leftarrow (\emptyset, F(\emptyset)) \).
2. for all \( i = 1, \ldots, \max \) do
3.   Using history \( h_{i-1} \), compute \( P_i \), the distribution over \( \Omega \).
4.   Sample element \( \omega_i \) according to \( P_i \).
5.   Set \( h_i \leftarrow (\omega_i, F(\omega_i)) \).
6. end for
7. if an element of \( T \) is contained in any tuple of \( h \) then
9. else
11. end if

Our framework is sufficiently general to apply to supervised learning, genetic algorithms, and sequential model-based hyperparameter optimization, yet is specific enough to allow for precise quantitative statements. Although faced with an inherent balancing act whenever formulating a new framework, we emphasize greater generality to allow for the largest number of domain specific applications.

IV. MEASURING PERFORMANCE

Since general search algorithms may vary the total number of sampling steps performed, we measure performance using the expected per-query probability of success,

\[
q(T, F) = \mathbb{E}_{P, h} \left[ \frac{1}{|P|} \sum_{i=1}^{|P|} P_i(\omega \in T) \bigg| F \right],
\]

(2)
where \( \tilde{P} \) is the sequence of probability distributions for sampling elements (with one distribution \( P_i \) for each time step \( i \)), \( H \) is the search history, and the \( T \) and \( F \) make the dependence on the search problem explicit. Because this is an expectation over sequences of history tuples and probability distributions, past information forms the basis for \( q(T,F) \). \( |\tilde{P}| \) denotes the length of the sequence \( \tilde{P} \), which equals the number of queries taken. The expectation is taken over all sources of randomness, which includes randomness over possible search histories and any randomness in constructing the various \( P_i \) from \( R_{0,i-1} \) (if such a construction is not entirely deterministic). Taking the expectation over all sources of randomness is equivalent to the probability of success for samples drawn from an appropriately averaged distribution \( \tilde{P}(\cdot | F) \) (see Lemma 1 of Appendix [16]). Because we are sampling from a fixed (after conditioning and expectation) probability distribution, the expected per-query probability of success for the algorithm is equivalent to the induced amount of probability mass allocated to target elements. Revisiting our fictional security scenario, the probability of capturing an enemy agent is proportional to the amount of security at his location. In the same way, the expected per-query probability of success is equal to the amount of induced probability mass at locations where target elements are placed. Thus, each \( \tilde{P}(\cdot | F) \) demarks an equivalence class of search algorithms mapping to the same averaged distribution; we refer to these equivalence classes as search strategies.

We use uniform random sampling with replacement as our baseline search algorithm, which is a simple, always available strategy and we define \( p(T,F) \) as the per-query probability of success for that method. In a natural way, \( p(T,F) \) is a measure of the intrinsic difficulty of a search problem [14], absent of any side-information. The ratio \( p(T,F)/q(T,F) \) quantifies the improvement achieved by a search algorithm over simple random sampling. Like an error quantity, when this ratio is small (i.e., less than one) the performance of the algorithm is better than uniform random sampling, but when it is larger than one, performance is worse. We will often write \( p(T,F) \) simply as \( p \), when the target set and information resource are clear from the context.

V. MAIN RESULTS

We state our main results here with brief discussion, and with proofs given in the Appendix (available online [16]).

A. Famine of Forte

**Theorem 1. (Famine of Forte) Define**

\[
\tau_k = \{ T \mid T \subseteq \Omega, |T| = k \in \mathbb{N} \}
\]

and let \( B_m \) denote any set of binary strings, such that the strings are of length \( m \) or less. Let

\[
R = \{ (T,F) \mid T \in \tau_k, F \in B_m \}, \quad \text{and} \quad R_{q_{\min}} = \{ (T,F) \mid T \in \tau_k, F \in B_m, q(T,F) \geq q_{\min} \},
\]

where \( q(T,F) \) is the expected per-query probability of success for algorithm \( \mathcal{A} \) on problem \((\Omega,T,F)\).

Then for any \( m \in \mathbb{N} \),

\[
\frac{|R_{q_{\min}}|}{|R|} \leq \frac{p}{q_{\min}}
\]

and

\[
\lim_{m \to \infty} \frac{|R_{q_{\min}}|}{|R|} \leq \frac{p}{q_{\min}}
\]

where \( p = k/|\Omega| \).

We see that for small \( p \) (problems with sparse target sets) favorable search problems are rare if we desire a strong probability of success. The larger \( q_{\min} \), the smaller the proportion. In many real-world settings, we are given a difficult search problem (with minuscule \( p \)) and we hope that our algorithm has a reasonable chance of achieving success within a limited number of queries. According to this result, the proportion of problems fulfilling such criteria is also minuscule. Only if we greatly relax the minimum performance demanded, so that \( q_{\min} \) approaches the scale of \( p \), do such accommodating search problems become plentiful.

This result has many applications, with some given in Section VI.

B. Conservation of Information

**Corollary 1. (Conservation of Active Information of Expectations) Define**

\[
I_{q(T,F)} = -\log p(T,F) \quad \text{and let}
\]

\[
R = \{ (T,F) \mid T \in \tau_k, F \in B_m \}, \quad \text{and} \quad R_{b} = \{ (T,F) \mid T \in \tau_k, F \in B_m, I_{q(T,F)} \geq b \}.
\]

Then for any \( m \in \mathbb{N} \)

\[
\frac{|R_{b}|}{|R|} \leq 2^{-b}.
\]

The active information transform quantifies improvement in a search over a baseline search method [14], measuring gains in search performance in terms of information (bits). This allows one to precisely quantify the proportion of favorable functions in relation to the number \( b \) bits improvement desired. Active information provides a nice geometric interpretation of improved search performance, namely, that the improved search method is equivalent to a uniform random sampler on a reduced search space. Similarly, a degraded-performance search is equivalent to a uniform random sampler (with replacement) on a larger search space.

Applying this transform, we see that finding a search problem for which an algorithm effectively reduces the search space by \( b \) bits requires at least \( b \) bits, so information is conserved in this context. Assuming you have no domain knowledge to guide the process of finding a search problem for which your search algorithm excels, you are unlikely to stumble upon one under uniform chance; indeed, they are exponentially rare in the amount of improvement sought.

C. Famine of Favorable Strategies

**Theorem 2. (Famine of Favorable Strategies) For any fixed search problem \((\Omega,t,f)\), set of probability mass functions \( \mathcal{P} = \{ P : P \in \mathbb{R}^{|\Omega|}, \sum_j P_j = 1 \} \), and a fixed threshold \( q_{\min} \in [0,1] \),**

\[
\frac{\mu(G_{t,q_{\min}})}{\mu(\mathcal{P})} \leq \frac{p}{q_{\min}},
\]

where \( G_{t,q_{\min}} = \{ P : P \in \mathcal{P}, t^TP \geq q_{\min} \} \) and \( \mu \) is Lebesgue measure. Furthermore, the proportion of possible search strategies giving at least \( b \) bits of active information of expectations is no greater than \( 2^{-b} \).

Thus, not only are favorable problems rare, so are favorable strategies. Whether you hold fixed the algorithm and try to match a problem to it, or hold fixed the problem and try to match a strategy to it, both are provably difficult. Because matching problems to algorithms is hard, seemingly serendipitous agreement between the two calls for further explanation, serving as evidence against blind matching by independent mechanisms (especially for very sparse targets embedded in very large spaces). More importantly, this result places hard quantitative constraints (similar to minimax bounds in statistical learning theory) on information costs for automated
machine learning, which attempts to match learning algorithms to learning problems [17], [18].

D. Success Under Dependence

**Theorem 3. (Success Under Dependence)** Define

\[ q = E_{T,F} [q(T, F)] \]

and note that

\[ q = E_{T,F} [P(\omega \in T|F)] = Pr(\omega \in T; A). \]

Then,

\[ q \leq \frac{I(T; F) + D(P_T||U_T)}{I_T} + 1 \]

where \( I_T = -\log k/|\Omega| \), \( D(P_T||U_T) \) is the Kullback-Leibler divergence between the marginal distribution on \( T \) and the uniform distribution on \( T \), and \( I(T; F) \) is the mutual information between \( T \) and \( F \). Theorem 3 provides a generalization bound that can be used to analyze the expected probability of success for the algorithm.

Thus, the bound on expected probability of success improves monotonically with the amount of dependence between target sets and information resources. We quantify the dependence using mutual information under any fixed joint distribution on \( x_k \) and \( B_m \). We see that \( I_T \) measures the relative target sparseness of the search problem, and can be interpreted as the information cost of locating a target element in the absence of side-information. \( D(P_T||U_T) \) is naturally interpreted as the predictability of the target sets, since large values imply the probable occurrence of only a small number of possible target sets. The mutual information \( I(T; F) \) is the amount of exploitable information the external resource contains regarding \( T \); lowering the mutual information lowers the maximum expected probability of success for the algorithm. Lastly, the 1 in the numerator upper bounds the contribution of pure randomness, as explained in the Appendix [16]. Thus, this expression constrains the relative contributions of predictability, problem difficulty, side-information, and randomness for a successful search, providing a satisfyingly simple and interpretable upper bound on the probability of successful search.

VI. EXAMPLES

A. Binary Classification

It has been suggested that machine learning represents a type of search through parameter or concept space [19]. Supporting this view, we can represent binary classification problems within our framework as follows:

- \( A \) - classification algorithm, such as an SVM.
- \( \Omega \) - space of possible concepts over an instance space.
- \( T \) - Set of all hypotheses with less than 10% classification error on test set, for example.
- \( F \) - set of training examples.
  - \( F(\emptyset) \) - full set of training data.
  - \( F(c) \) - loss on training data for concept \( c \).
- \( (\Omega, T, F) \) - binary classification learning task.

The space of possible binary concepts is \( \Omega \), with the true concept being an element in that space. In our example, let \( |\Omega| = 2^{100} \). The target set consists of the set of all concepts in that space that (1) are consistent with the training data (which we will assume all are), and (2) differ from the truth in at most 10% of positions on the generalization hold-out dataset. Thus, \( |T| = \sum_{i=0}^{10} \binom{100}{i} \). Let us assume the marginal distribution on \( T \) is uniform, which isn’t necessary but simplifies the calculation. The external information resource \( F \) is the set of training examples. The algorithm uses the training examples (given by \( F(\emptyset) \)) to produce a distribution over the space of concepts; for deterministic algorithms, this is a degenerate distribution on exactly one element. A single query is then taken (i.e., a concept is output), and we assess the probability of success for the single query. By Theorem 3, the expected chances of outputting a concept with at least 90% generalization accuracy is thus no greater than \( \frac{I(T; F) + D(P_T||U_T)}{I_T} \leq \frac{1}{2^{99}} \). The denominator is the information cost of specifying at least one element of the target set and the numerator represents the information resources available for doing so. When the mutual information meets (or exceeds) that cost, success can be ensured for any algorithm perfectly mining the available mutual information. When noise reduces the mutual information below the information cost, the expected probability of success becomes strictly bounded in proportion to that ratio.

B. General Learning Problems

Vapnik presents a generalization of learning that applies to classification, regression, and density estimation [20], which we can translate into our framework. Following Vapnik, let \( P(z) \) be defined on space \( Z \), and consider the parameterized set of functions \( Q_\alpha(z) \), \( \alpha \in \Lambda \). The goal is to minimize \( R(\alpha) = \int Q_\alpha(z) dP(z) \) for \( \alpha \in \Lambda \), where \( P(z) \) is unknown but an i.i.d. sample \( z_1, \ldots, z_\ell \) is given. Let \( R_{emp}(\alpha) = \frac{1}{\ell} \sum_{i=1}^{\ell} Q_\alpha(z_i) \) be the empirical risk.

To reduce this general problem to a search problem within our framework, assume \( \Lambda \) is finite, choose \( \epsilon \in \mathbb{R}_{\geq 0} \), and let

- \( \Omega = \Lambda \);
- \( T = \{ \alpha : R(\alpha) - \min_{\alpha' \in \Lambda} R(\alpha') < \epsilon \} \);
- \( F = \{ z_1, \ldots, z_\ell \} \);
- \( F(\emptyset) = \{ z_1, \ldots, z_\ell \} \);

and

\[ F(\alpha) = R_{emp}(\alpha). \]

Thus, any finite problem representable in Vapnik’s statistical learning framework is also directly representable within our search framework.

C. One-Size-Fits-All Fitness Functions

Theorem 1 gives an upper bound on the proportion of favorable search problems, but what happens when we have a single, fixed information resource, such as a single fitness function? A natural question to ask is for how many target locations can such a fitness function be useful. More precisely, for a given search algorithm, for what proportion of search space locations can the fitness function raise the expected probability of success, assuming a target element happens to be located at one of those spots?

Applying Theorem 1 with \( |T| = 1 \), we find that a fitness function can significantly raise the probability of locating target elements placed on, at most, \( 1/q_{min} \) search space elements. We see this as follows. Since \( |T| = 1 \) and the fitness function is fixed, each search problem maps to exactly one element of \( \Omega \), giving \( |\Omega| \) possible search problems. The number of \( q_{min} \)-favorable search problems is upper-bounded by

\[ \frac{|\Omega|}{q_{min}} \leq \frac{|\Omega|}{q_{min}} = \frac{1}{q_{min}}. \]

Because this expression is independent of the size of the search space, the number of elements for which a fitness function can strongly raise the probability of success remains fixed even as the size of the search space increases. Thus, for very large search spaces the proportion of favored locations effectively vanishes. There can exist no single fitness function that is strongly favorable for many elements simultaneously, and thus no “one-size-fits-all” fitness function.
Our results also help make sense of recent trends in machine learning. They explain why new algorithms continue to be developed, year after year, conference after conference, despite decades of research in machine learning and optimization. Given that algorithms can only perform well on a narrow subset of problems, we must either continue to devise novel algorithms for new problem domains or else move towards flexible algorithms that can modify their behavior solely through parameterization. The latter effectively behave like new strategies for new hyperparameter configurations. The rise of flexible, hyperparameter sensitive algorithms like deep learning methods and vision architectures shows a trend towards the latter, with their hyperparameter sensitivity being well known [21], [22].

VII. CONCLUSION

A colleague once remarked that the results presented here bring to mind Lake Wobegone, “where all the children are above average.” In a world where every published algorithm is “above average,” conservation of information reminds us that this cannot be. Improved performance over any subset of problems necessarily implies degraded performance over the remaining problems [10], and all methods have performance equivalent to random sampling when uniformly averaged over any closed-under-permutation set of problems [1], [2], [4].

But there are many ways to achieve an average. An algorithm might perform well over a large number of problems, only to perform poorly on a small set of remaining problems. A different algorithm might perform close to average on all problems, with slight variations around the mean. How large can the subset of problems with improved performance be? We show that the maximum proportion of search problems for which an algorithm can attain q-favorable performance is bounded from above by p/qmax. Thus, if an algorithm greatly excels on a class of problems, the size of that class must be relatively small, since an algorithm can only perform well over a narrow subset of possible problems. Not only is there no free lunch, but there is also a famine of favorable problems.

If finding a good search algorithm for a fixed algorithm is hard, then so is finding a good search algorithm for a fixed problem (Theorem 2). Thus, the matching of problems to algorithms is provably difficult, regardless of which is fixed and which varies.

Our results paint a more optimistic picture once we restrict ourselves to those search problems for which the external information resource is strongly informative of the target set, as is often assumed to be the case. For those problems, the single-query probability of success is upper bounded by a function involving the mutual information between target sets and external information resources (like training datasets and objective functions). The lower the mutual information, the lower chance of success, but the bound improves as the dependence is strengthened.

The search framework we propose is general enough to find application in many problem areas, such as machine learning, evolutionary search, and hyperparameter optimization. The results are not just of theoretical importance, but help explain real-world phenomena, such as the need for exploitable dependence in machine learning and the empirical difficulty of automated learning [18]. Our results help us understand the growing popularity of deep learning methods and unavoidable interest in automated hyperparameter tuning methods. Extending the framework to continuous settings and other problem areas (such as active learning) is the focus of ongoing research.

REFERENCES