Information Perspective of Optimization

Yossi Borenstein and Riccardo Poli

University of Essex, Colchester, UK
{yboren, rpoli}@essex.ac.uk

Abstract. In this paper we relate information theory and Kolmogorov Complexity (KC) to optimization in the black box scenario. We define the set of all possible decisions an algorithm might make during a run, we associate a function with a probability distribution over this set and define accordingly its *entropy*. We show that the expected KC of the *set* (rather than the function) is a better measure of problem difficulty. We analyze the effect of the entropy on the expected KC. Finally, we show, for a restricted scenario, that any permutation closure of a single function, the finest level of granularity for which a No Free Lunch Theorem can hold [7], can be associated with a particular value of entropy. This implies bounds on the expected performance of an algorithm on members of that closure.

1 Introduction

General purpose algorithms (also known as metaheuristics, black-box algorithms or randomized search heuristics [9]) are often used when either the problem is not well defined or when there is insufficient knowledge (or resources) to construct specific algorithms [9]. In the most general case, a randomized search heuristic can be represented as a mapping from a multi-set of previously visited points to a new (not necessarily unvisited) point in the search space. Wolpert and Macready [10], Vose [8] as well as Droste, Jansen and Wegener [3,9] suggested accordingly a formal model for black-box algorithms which generalizes most, if not all, existing randomized search heuristics.

There are various theoretical approaches to analyze the expected performance of metaheuristics. Wolpert and Macready [10], using their model, proved that over all possible problems, all algorithms have the same performance. It was later shown that the same result holds even for a smaller set: the sharpened No Free Lunch Theorem (NFLT) [7] proves that a NFL result holds for any set of functions which is closed under permutation.

Rather than focusing on a set of functions, compressibility or Kolmogorov complexity is associated with a particular object. It is defined as the length of the shortest program that can generate a string and halts. When the string represents a problem, it is argued that compressible strings are associated with easy problems whereas random, incompressible strings, with difficult ones [6].

In [2], based on the information landscape framework, we derived a new way to measure the KC of a fitness function and showed the connection to the NFLTs. However, this framework was based, mainly, on a first order approximation. This paper removes this restriction. In section 2 we introduce the notion of Kolmogorov complexity – a way to measure the complexity of a fitness function f. Section 3 defines the information

T.P. Runarsson et al. (Eds.): PPSN IX, LNCS 4193, pp. 102–111, 2006.
© Springer-Verlag Berlin Heidelberg 2006

content of f. Then, we define a way to measure its entropy, or as we put it, to quantify the amount of information a function contains. The connection between the expected difficulty of the function and its amount of information is explored in section 4. The implication of this on the sharpened NFLTs is given in section 5.

2 Kolmogorov Complexity

The Kolmogorov complexity [6] $K : \{0,1\}^* \rightarrow \mathbb{N}$ is a function from finite binary strings of arbitrary length to the natural numbers $\mathbb{N}$. It is defined on 'objects' represented by binary strings but can be extended to other types like functions.

The KC of an object, x, is defined as the length of the shortest program that prints x and halts. The program can be implemented by any universal programming language (e.g., C++, Java), in which a universal Turing Machine can be implemented. The choice of universal computer (programming language) may change the KC of x only by a fixed additive constant (which depends on the length of code required to simulate a universal computer by another). For this reason, we fix our programming language to an arbitrary language (e.g., C) and define KC w.r.t. that language. The KC of a string x is defined as: $K(x) := min_p\, l(p)$, where p is a program that prints x and halts and $l(p)$ denotes the length of program p. This definition suffices for the purpose of this paper – a more accurate definition is given in [6,5].

A simple or regular object x has a low KC. For example, a string representing a sequence consisting of n 1s (i.e., "111...1") can be represented by $K(x) = O(\log n)$ bits. The size of the program depends on the number of bits needed to encode the number n. A random object, on the other hand, with very high probability, can only be represented by $K(x) = n + O(\log n)$. That is, the shortest program to print x will be: *"print x"*. The KC of an object can only be used as a *conceptual* measure of its complexity, *it cannot be computed*. That is, it is possible to analyze the properties of a random object, or even to prove that the majority of objects are random but, given an object x, it is not possible to prove that it is random.

The notion of KC can be extended to account for functions as well. Let $f : X \rightarrow Y$ be a mapping between two finite spaces X and Y, then:

$$K(f) = min_{p_f \in \{0,1\}^*} \{l(p_f) : \forall x \in X\ p_f(x) = f(x)\}$$

where p_f is a program that given an input x returns the value $f(x)$. Any function, in the worst case, can be represented by explicitly listing the co-domain value for each domain (e.g.,*if (x="0000000000") return 1230 else...*). If the function is random, this is the *only* way to represent it. Other functions, like flat functions, can be represented in code with a constant length, e.g., *"return 0"*.

The KC of a function is sometimes used as an indicator for the expected difficulty of the function for optimization algorithms. A random function contains no regularities. For this reason, no algorithm, regardless of the search strategy, is expected to optimize it efficiently. A function with a low KC, on the other hand, contains regularities, that, in some scenarios, can be exploited by a search algorithm to find an optimal solution quickly.

Some limitation of using KC to asses problem difficulty were studied in [1]. It was concluded that KC measures how different (either better or *worse*) the expected performance (over a function) is likely to be from a random search. Moreover, some examples of difficult functions which have low KC were given. Nearly constant functions, like the needle-in-a-haystack, have minimal KC. Nevertheless, they are very difficult to optimize. This paper addresses particularly the last limitation. In this paper we make the following assumptions: Firstly, we focus, on functions which contain only a *small* number of optima. More precisely, by small, we will mean logarithmic in the size of the search space. Secondly, we assume that the algorithm has no a priori bias towards specific regions of the search space. We do not exclude deterministic decisions making – we assume, however, that the initial starting points are random. In the following section, we will define the *information content* of a function, and we will use this in section 4 to suggest a new way to calculate the KC.

3 Information Content of a Function

Let $f : X \to Y$, where X denotes a finite search space and Y is finite. Let F denote all possible fitness functions. Using Vose's notation [8], random search heuristics can be thought of as an initial collection of elements $\Psi_k \in \Psi$ chosen from some search space X together with a transition rule τ which produces from the collection Ψ_k another collection Ψ_l. The search is a sequence of iterations of τ: $\Psi_k \xrightarrow{\tau} \Psi_l \xrightarrow{\tau} \cdots$. A collection of elements is a multiset of X. We use the term *search-state* to denote a particular collection. The set of all possible such collections, the state-space, is denoted by Ψ. Without loss of generality, we assume a notion of order in Ψ. Note that we do not consider the dynamics of the algorithm and hence adjacent states do not correspond to adjacent time steps.

We restrict our attention to search algorithms for which τ depends completely on the current state, that is: $\tau : \Psi \times F \to \Psi$. Heuristics such as particle swarm optimization include other parameters such as, for example, velocity vectors. We do not consider, at this stage, such algorithms.

In reality, the transition rule $\tau(\Psi_i, f)$ if often a composition $\tau = \chi \circ \xi(\Psi_i, f)$ where $\xi(\Psi_i, f)$ denotes a *selection* operator, and χ can be thought of as the *exploration* operator. The selection phase identifies solutions with high fitness value, the exploration operator samples accordingly new solutions from X. *This section focuses solely on the selection phase of the search.*

In order to make a clear distinction between the two operators (stages) it is useful to think of an output of the selection operator, a multiset, d, which represents a possible way to choose (or select) solutions from Ψ_i. For example, in GAs, given a particular population, Ψ_i, d includes a possible mating pool. For a (1+1) evolutionary strategy, d can be either the parent or the offspring[1]. Given a state Ψ_i, we denote by S^i all possible ways of selection points. That is, S^i is a set of multisets, each multiset, $d \in S^i$, corresponds to one possible way of selecting points.

[1] The notion of a state, in that case, is not natural but, nevertheless, correct. Each state, in our notation, contains *two* solutions, then selection is applied to select one of them, and a mutation is applied in order to achieve the next state.

The dependency of the performance of a search algorithm on f is reflected by a probability distribution, P_f^i, that the selection mechanism defines for each state over S^i. The particular multiset of solutions, d, that the algorithm selects, being the only argument for the exploration operator, *defines* the next state of the search. We define the *information content* of f as the set of all such distributions.

Definition 1. *The information content of the function f is the set $\mathcal{P}_f = \{P_f^1, P_f^2, ..., P_f^n\}$ which gives for each state, Ψ_i the probability distribution P_f^i used in the selection phase.*

Usually, the algorithm does not define explicitly a probability distribution over S^i, rather, a distribution over single solutions from Ψ_i. For example, binary tournament selection defines the probability of selecting one of two possible solutions as follows:

$$\Pr_{\text{trnmnt}}\{x \mid \{x, y\}\} = \delta(f(x) > f(y)) + 0.5\delta(f(x) = f(y)) \tag{1}$$

where the function $\delta(\texttt{expr})$ returns 1 if `expr` is true, and 0 otherwise. This is used for a state (population) bigger than two points, by selecting, iteratively, uniformly at random, two points from Ψ_i and applying equation 1:

$$\Pr(x \mid \Psi_i, f) = \Pr\{x, x\} + \sum_{x \neq y} \Pr\{x, y\} \cdot \Pr_{\text{trnmnt}}\{x \mid \{x, y\}\} \tag{2}$$

Finally, P_f^i, the probability of selecting a particular multiset, d, is obtained as follows:

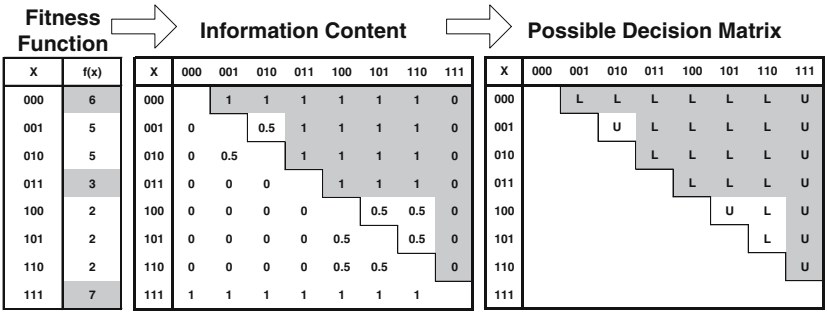
$$P_f^i(d \mid \Psi_i, f) = \prod_{j<|d|} \Pr(d_j \mid \Psi_i, f). \tag{3}$$

Rather than calculating the probability of selecting a particular multiset on a particular state ($P_f^i(d)$), we can measure the probability of obtaining particular sequence of decisions – this gives a full account (when, to reiterate, all the other parameters of the algorithm are fixed) for the expected performance. Let $\mathbb{D} = S^0 \times S^1 \times \cdots \times S^n$, and $D \in \mathbb{D}$ a particular decision set. Let R be a random variable taking values from $\mathbb{D}$. The probability $\mathcal{P}_f$ that the algorithm is consistent with D is:

$$\mathcal{P}_f(R = D) = \prod_{d \in D} P_f^i(d)$$

In order to understand the meaning of the distribution $\mathcal{P}_f$, it is important to explain the connection between a deterministic selection mechanism, decision set and fitness functions. For deterministic selection mechanism, f and D are synonymous: there is only one decision set which corresponds to a particular fitness function. The set of all possible fitness functions (F) corresponds therefore to a set of possible decision sets $\mathcal{D} \subset \mathbb{D}$. For stochastic selection mechanisms, a uniform P_f corresponds to choosing at each state, Ψ_i, $d \in S^i$ uniformly, at random, this can be thought of as running a deterministic search algorithm BUT choosing $f \in F$ uniformly, at random.[2] The NFLTs

[2] Assuming that the algorithm is consistent, i.e., given the same state, it always makes the same decision.



X	f(x)
000	6
001	5
010	5
011	3
100	2
101	2
110	2
111	7

X	000	001	010	011	100	101	110	111
000		1	1	1	1	1	1	0
001	0		0.5	1	1	1	1	0
010	0	0.5		1	1	1	1	0
011	0	0	0		1	1	1	0
100	0	0	0	0		0.5	0.5	0
101	0	0	0	0	0.5		0.5	0
110	0	0	0	0	0.5	0.5		0
111	1	1	1	1	1	1	1	

X	000	001	010	011	100	101	110	111
000		L	L	L	L	L	L	U
001			U	L	L	L	L	U
010				L	L	L	L	U
011					L	L	L	U
100						U	L	U
101							L	U
110								U
111								

Fig. 1. Matrix representation of the information content of a function and a possible decision set

[7] imply that the expected performance, in such case, is that of a random search. The distribution $\mathcal{P}_f$ corresponds therefore to the level of randomness in the decision making of the algorithm. We define the entropy of f, in order to measure this[3]:

Definition 2. *Let Ψ denote the state-space of the algorithm. The entropy $H(f)$ of f corresponds to the entropy of P_f as defined by the algorithm.*

$$H(f) \equiv \sum_{D \in \mathbb{D}} P_f(D) \log 1/P_f(D)$$

While $K(f)$ measures how regular a function is, and thus implies how difficult it is to optimize the function, $H(f)$ measures the hardness of the search from a different perspective. It explicitly measures the randomness induced by the way an algorithm is using the function. The higher the rate of random decisions the algorithm is expected to make, the closer the performance will be to a random search.

3.1 Making It Concrete

The size of $|\Psi|$ and $|\Psi_i|$ for realistic search algorithms is usually bounded. Genetic algorithms usually use a population of fixed size, the size of a tabu-list is bounded and local-searchers often consider only two solutions at any given time. In order to have a more concrete formulation we will restrict our attention to algorithms that use a comparison of pairs of solutions in the search space.

Each state Ψ_i corresponds, therefore, to a pair of solutions (e.g., $\{x_i, x_j\}$) from which the algorithm chooses one. It follows that the number of all possible states $|\Psi| = |X|^2$, and that for each state there are $|S_i| = 2$ possible choices. In this case, if we consider various local searchers, equations 1 and 2 coincide, and therefore:

$$P_f^i(\{x\} \mid \{x, y\}) = \Pr_{\text{trnmnt}}\{x \mid \{x, y\}\} \tag{4}$$

[3] In order to have a complete understanding one has to consider the probability of obtaining a particular state during a run and calculate the expected entropy accordingly. At this stage, we assume that the uniform distribution gives a good indication to that.

Figure 1 illustrates this notion for a search space of size 8. It represents a fitness function f, the information content of the function and a possible decision set. The information content of the function is represented in the form of a matrix, in which the area above the diagonal corresponds to the search-states Ψ. Each entry in the matrix corresponds to the probability P_f^i, defined by equation 4 of choosing either of the $|S^i| = 2$ possible solutions. The decision set is defined as a possible realization of the distributions given in the information-content matrix. We used the abbreviation 'U' and 'L' to denote up and left, respectively, that is, to point the particular solutions in the matrix, which were selected.

This scenario corresponds directly to various kinds of local search algorithms. However, it can also *approximate* some population-based search algorithms, as long as they use a binary selection operator. For example, a GA with binary tournament selection, or even Memetic algorithms. *In any case, from this point on, we will consider only this restricted scenario*. Whenever Ψ is implied we assume the comparison of pairs of solutions. P_f^i is defined as in equation 4. Under this assumptions we can calculate the entropy of an algorithm.

Theorem 1. *Let X, Ψ, f, P_f^i defined as before. Define $r(f) = \sum_i \delta(P_f^i = 0.5)$. Then, $H(f) = r(f)$.*

Proof. Since for $P_i^f = 0.5$ the algorithm chooses with equal probability one of the two solutions, $P_f(D) = \prod_{d \in D} P_f^i(d)$ is uniformly distributed. Also, the number of possible decision sets is exponential with the number of entries $P_i^f = 0.5$. It follows, therefore, that $\forall_D \; P_f(D) = 1/2^{r(f)}$ which gives, $H(f) \equiv \sum_{D \in \mathbb{D}} 1/2^{r(f)} \log 2^{r(f)} = r(f)$ □

The entropy of the landscape corresponds to the number of states in which the algorithm is forced to take a random decision. The entropy will be maximal (i.e., equals $|\Psi|$) for a flat landscape (in which $\forall_i P_f^i = 0.5$). In this case, the search will be random. Note that while low entropy indicates non-random search, this does not necessarily correspond to efficient search – it simply implies that the algorithm searches according to a certain bias. The efficiency of the search depends on the matching between this bias and the problem at hand.

4 Information and Problem Hardness

In section 2 we defined the KC of f and argued that high KC implies hardness. Section 3 on the other hand, focused on the entropy of f for stochastic search algorithms. It seems that the two notions define hardness from two different perspectives: the first, $K(f)$, relates to some intrinsic property of f which measures how regular the function is. The second, $H(f)$, measures the level of randomness as reflected by the way an algorithm uses f. In this section we show that the KC of the expected *decision set* of the algorithm, integrates these two measures.

As a first step, note that given the fitness function f, we can use a fixed size program to generate the information content of f (i.e., equation 4). This, in turn, can be used to generate all the elements of the decision set which correspond to entries with a value

different from 0.5. The KC of the decision vector is equivalent therefore to that of f plus the complexity which corresponds to the random decisions the algorithm makes.

The problem arises as to how to measure the part corresponding to the random decisions. It seems that it can be better described by $H(f)$ (which relates to the distribution) rather than the Kolmogorov complexity which measures the complexity of a single object. The two measurements of information, however, are closely related. The following result is taken from [6]:

Lemma 1. *Let $x = x_1x_2...$ where the individual x_i are realizations of some random variable X_i, distributed according to some distribution P. If all outcomes $X_1, X_2, ...$ are independently identically distributed (i.i.d.) with for all i, $P(X_i = 1) = p$ for some $p \in [0, 1]$, the expected Kolmogorov complexity of x is:*

$$K(x_{[1:n]}) = n \cdot H(p) + o(n) \tag{5}$$

where $H(p) = -p \log p - (1 - p) \log(1 - p)$ is the binary entropy such that $0 \leq H(p) \leq 1$.

Lemma 1 makes the connection clear. Since the elements in the decision set which correspond to the random decision the algorithm makes are i.i.d., equation 5 gives us the expected KC for such a string. The following is a simple corollary of that.

Corollary 1. *Let X, Ψ, f, P_f^i defined as before. Let D denote the expected decision set. $K(D) > r(f)$*

Proof. The decision set D can be decomposed into two subsets: let $D^1 = \{d|P_f^i(d) \neq 0.5\}$ and $D^{0.5} = \{d|P_f^i(d) = 0.5\}$. Clearly, $K(D|f) = K(D^{0.5}) + O(1)$ (equation 4 and a simple loop can generate, given f the subset D^1). But, following lemma 1, $K(D^{0.5}) = |D^{0.5}| \cdot H(0.5) + o(|D^{0.5}|) = r(f) \cdot 1 + o(r(f))$. Which gives: $K(D) \geq K(D|f) > r(f)$ □

In order to understand the implication of this lemma, consider the expected hardness of a needle-in-a-haystack (NIAH). The NIAH describes a landscape in which all the points in the search space have the same fitness except the global optimum which has a higher one. It is known to be a very difficult problem.

For a search space of size n let f_{needle} denote the NIAH and $\mathcal{P}_f$ the corresponding information content. The average description length of f_{needle} (and therefore of $\mathcal{P}_f$ as well) is $O(\log n)$. The high compressibility of the function in contrast to its known hardness is usually given as a counterexample to the use of Kolmogorov complexity as a measure of problem difficulty [1].

This apparent contradiction can be resolved by considering the algorithm perspective of the NIAH landscape. The algorithm does not see a flat landscape. At each time step it has to make a concrete decision, namely to decide which solution to sample next. It therefore has to "interpret" the flat landscape to a landscape which contains concrete information. In other words, it selects uniformly, at random $D \in \mathbb{D}$ and selects solutions accordingly.

Following the previous lemma consider the Kolmogorov complexity of the expected decision set vs. the fitness function:

$$K(D_{needle}) \geq r(f_{needle}) = 2^{((n-1)\cdot(n-2))/2} \gg \log n$$

This illustrates the magnitude of difference between the current approach to calculate the Kolmogorov complexity of a landscapes and the one suggested in this paper. The NIAH, however, is an extreme example. Generally, moving from low entropy ($H(f) = 0$) to maximum entropy ($H(f) = |\Psi|$) we should obtain many intermediate values.

5 Information and the Sharpened NFLT

In the previous section we showed that $K(D_f) > H(f)$, where D_f is the expected decision set, given the function f. Interestingly, $H(f)$ depends on the *fitness distribution* of f alone. This allows us to make an important observation regarding the sharpened NFLT [7]. In the following we give a brief summary of the sharpened NFLT and then show how it is connected with our results.

Let $f : X \rightarrow Y$ be a function and $\sigma : X \rightarrow X$ be a permutation (i.e., σ is one-to-one and onto). The permutation σf of f is the function $\sigma f : X \rightarrow Y$ defined by $\sigma f(x) = f(\sigma^{-1}(x))$.

Define a set F of functions to be closed under permutation if for every $f \in F$, every permutation of f is also in F. Schumacher, Vose and Whitley [7] proved that the permutation closure of a single function is the finest level of granularity at which a NFL result can hold.

English [4] named each set F, a block. A distribution which is uniform within each block is called block uniform. The space of all possible problems can be divided into blocks. A block uniform distribution is necessary and sufficient for NFLT in search.

Each block (i.e., a set F) can be associated with a particular value of entropy. For each $f \in F$, $H(f)$ is:

$$H(f) = \Sigma_i \binom{|y = i|}{2} \tag{6}$$

Equation 6 counts all possible pairs of solutions which have the same fitness. It follows that the entropy depends solely on the *fitness distribution* of f. Since the permutation operator does not affect the fitness distribution of a function, all the functions which belong to F have the same fitness distribution and, therefore, the same entropy, this proves the following result:

Theorem 2. *Let F be a set of functions which is c.u.p. $\forall_{f,g \in F}\ H(f) = H(g)$. We denote the entropy of F, $H(F) = H(f \mid f \in F)$*

5.1 A *Qualitative* Plot of the Expected Hardness of a Problem

Let F be c.u.p. defined over the metric space (X, d). In this section we would like to analyze how the variance of expected performances changes for different values of $H(F)$. Let us start with $H(F) \approx |\Psi|$. It follows from corollary 1 that $\forall_{f \in F} K(D_f) > |\Psi|$, this corresponds to a random function and hence, *all the functions* in the set are expected to be very difficult to optimize. The variance, however, is expected to be very low – irrespectively of the algorithm, the expected performance on *any* function is expected to be the same, that of a random search.

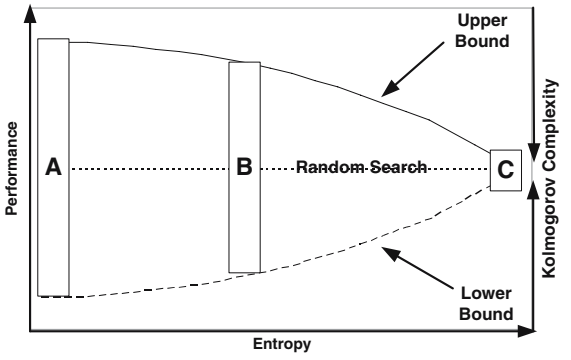


Fig. 2. The variance of expected performance as a function of the entropy. Each rectangle represents a block uniform on which a NFL result holds. The space of all possible problems is the union of all blocks. Each block is associated with a particular entropy. (A) represents a block with minimal entropy and maximum variance, (C) maximum entropy and no variance (B) is in between.

On the other hand, for $H(F) \approx 0$ we do not have any bound on the performance. We can choose, for example, a function f_1 such that $K(D_{f_1})$ is minimal. The low KC suggests that there exists an algorithm, a, which is *the best possible algorithm* to solve f. This implies that f is expected to be very easy to a, the question arises, though, as to how well the same algorithm performs on other functions from F. It is important to remember that KC gives an indication to the best performance that a realistic algorithm can have over one, specific function. It does not tell us how well the same algorithm may perform on other functions. From the sharpened NFLTs on the other hand, we know that for each problem (e.g., f_1) on which an algorithm (e.g., a) performs better than random search there exists another problem (e.g., f_2), in the same set, on which it performs as badly. It is easy to construct two functions $f_1, f_2 \in F_2$ such that one is expected to be very easy and the other very hard. For example, let:

$$f_1(x) = x$$

$$f_2(x) = \begin{cases} n+1 & \text{if } x = 0, \\ x & \text{otherwise} \end{cases}$$

We showed that for $H(F) = |\Psi|$ the variance of possible performance values is small, and, on the other hand, for $H(F) = 0$ the variance is high. More generally, moving from $H(F) = 0$ to $H(F) = |\Psi|$ the variance becomes smaller and smaller. The reason for that is derived directly from the NFLTs. An algorithm is expected to perform well on a problem (better than random search) *only if* it is aligned with the problem. The same algorithm, in that case, is misaligned with another problem and, therefore, is expected to perform badly (i.e., worse than a random search), to the same extent. The entropy $H(f)$ can be thought of as measuring how *unbiased* an algorithm is. The higher $H(f)$ the more random decisions the algorithm makes, the less bias it will be.

To summarize, corollary 1 shows that $K(D_f) > H(f)$: the entropy of a function is a lower bound to its expected KC. Theorem 2 associates each group of functions which is c.u.p with a particular entropy. Thus, each group of functions for which the NFLTs hold, are associated with a certain value of entropy which gives a bound on the expected performance of any algorithm when solving members of that group.

So, on one hand, given the entropy we have a qualitative boundary on the performance of the most efficient algorithm on the easiest landscape. On the other hand, from the sharpened NFLT we know that if the algorithm performs well on one problem there exists another problem on which it performs as badly. Combining these two aspects we can give a qualitative plot of the expected performance of the best algorithm over all possible problems w.r.t. their entropy. This is illustrated in figure 2.

6 Conclusion

This paper has provided a connection between information theory, Kolmogorov complexity and the study of optimization algorithms. The information content of a fitness function was defined, its expected effect on performance was analyzed and a novel way to compute the KC of a fitness function was introduced. This has led to new observations regarding the sharpened NFLTs: we proved that each closure can be associated with a particular entropy which implies bounds on its Kolmogorov complexity and consequently, expected difficulty. In order to make some of these connections we had to limit our attention to particular kind of algorithms (section 3.1). In future research, we plan to generalize our results.

References

1. Y. Borenstein and R. Poli. Kolmogorov complexity, optimization and hardness. CEC 2006.
2. Y. Borenstein and R. Poli. No free lunch, Kolmogorov Complexity and the information landscape. In *Proceedings of IEEE CEC 2005*, volume 3, 2005.
3. S. Droste, T. Jansen, and I. Wegener. Upper and lower bounds for randomized search heuristics in black-box optimization. *ECCC*, (048), 2003.
4. T. M. English. On the structure of sequential search: Beyond "no free lunch". In J. Gottlieb and G. R. Raidl, editors, *EvoCOP 2004*, volume 3004, pages 95–103. Springer, 2004.
5. P. Grunwald and P. Vitanyi. Shannon information and kolmogorov complexity. *IEEE Transactions on Information Theory*, 2004. In Review.
6. P. Grunwald and P. Vitanyi. Algorithmic information theory. In *Handbook on the Philosophy of Information*. Elsevier, to appear.
7. C. Schumacher, M. D. Vose, and L. D. Whitley. The no free lunch and problem description length. In L. Spector et al., editors, *GECCO-2001*, pages 565–570. Morgan Kaufmann.
8. M. D. Vose. *The Simple Genetic Algorithm: Foundations and Theory*. MIT Press, Cambridge, MA, USA, 1998.
9. I. Wegener. Towards a theory of randomized search heuristics. In B. Rovan and P. Vojtás, editors, *MFCS*, volume 2747, pages 125–141. Springer, 2003.
10. D. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Trans. Evolutionary Computation*, 1(1):67–82, 1997.