Decomposition of Fitness Functions in Random Heuristic Search

Yossi Borenstein and Riccardo Poli

Department of Computer Science, University of Essex, UK
{yboren,rpoli}@essex.ac.uk

Abstract. We show that a fitness function, when taken together with an algorithm, can be reformulated as a set of probability distributions. This set can, in some cases, be equivalently viewed as an information vector which gives ordering information about pairs of search points in the domain. Certain performance criteria definable over such an information vector can be learned by linear regression in such a way that extrapolations can sometimes be made: the regression can make performance predictions about functions it has not seen. In addition, the vector can be taken as a model of the fitness function and used to compute features of it like difficultly via vector calculations.

1 Introduction

Genetic algorithms (GAs) are problem independent heuristics which have been reported to perform relatively well on problems for which only partial knowledge is available. One of the main challenges of the field of evolutionary computation is to predict the behavior of GAs. In particular, the goal is to be able to classify problems as hard or easy according to the performance GA would be expected to have on such problems, before actually running the GA.

A first avenue in this direction was the Building Block (BB) hypothesis [8], which states that a GA tries to combine low, highly fit schemata. Following the BB hypothesis the notion of deception [8,6] isolation [9] and multimodality [19] have been defined. These were able to explain a variety of phenomena. Unfortunately, they did not succeed in giving a reliable measure of GA-hardness [10,11].

Given the connection between GAs and theoretical genetics, some attempts to explain the behavior of GAs were inspired by biology. For example, epistasis variance [4] and epistasis correlation [15] have been defined in order to estimate the hardness of a given real world problem. NK landscapes [2,13] use the same idea (epistasis) in order to create an artificial, arbitrary, landscape with a tunable degree of difficulty. These attempts, too, did not succeed in giving a full explanation of the behavior of a GA [11,16].

Finally, fitness distance correlation [12] tries to measure the intrinsic hardness of a landscape, independently of the search algorithm. Despite good success, fitness distance correlation is not able to predict performance in some scenarios [11].

The partial success of these approaches is not surprising. Several difficulties present themselves when developing a general theory that explains the behavior of a GA and is able to predict how it will perform on different problems.

A GA is actually a family of different algorithms. Given a problem, the GA designer first decides which representation (e.g. binary, multiary, permutation or real numbers) to use, then how to map the solution space into the search space, and finally which operator(s) (mutation, crossover) to use. Moreover, there are limited concrete guidelines on how to choose a representation and a genotype-phenotype mapping. Indeed this is a very difficult task. Different genotype-phenotype representations can completely change the difficulty of a problem [7]. There have been attempts to evolve the right representation [1] and there are some general design guidelines [14,18,7]. However, the reality is that the responsibility of coming up with good ingredients for a GA is still entirely on the GA designer.

In this paper we show that a fitness function, f, *when taken together with an algorithm*, can be reformulated as a vector of probability distributions, $\mathcal{P}_f$ (section 2). Following this decomposition, we are able to derive a first order approximation to the performance of Randomized Search Heuristics (RSHs). Using the GA as an example, we show that for small problems regression can be used to learn the coefficients of a linear model and predict the performance for unseen problems.

Using our construction, a measure of distance between two functions can be defined. We suggest to measure problem difficulty by measuring the distance between a function and the easiest possible function for an algorithm.

We measure the exploration ability of the algorithm as the expected entropy of the distribution that the selection mechanism defines (given a particular function) over the different search states. High values of entropy indicate that the expected performance of GA over the function is poor.

Due to the size of the sequence of probability distributions $\mathcal{P}_f$, the measurement suggested in section 2 cannot be used in practice. In section 3 we argue that $\mathcal{P}_f$ can (at least in some cases) be equivalently viewed as an information vector V, which gives ordering information about pairs of search points in the domain. While this is an approximation, it allows us to make concrete measurements and test our assumptions empirically. Section 3.1 suggests a first order approximation to the performance of RSHs on V. This is then tested empirically (section 4.1) on toy problems (V is much smaller than $\mathcal{P}_f$ however it is still too big). In section 3.3 we consider an indicator to the expected entropy defined in section 2.3. Using the first order approximation we show that the entropy may imply bounds on the expected performance. Finally, in section 3.2 we consider a vector V_{max} as the equivalent of the easiest fitness function. We argue that the distance between any other vector V and V_{max} can be used as an indication to problem difficulty. This is corroborated by experiments in section 4.2.

2 Algorithmic-Decomposition of the Fitness Function

Once all the parameters of a heuristic are chosen (i.e., representation, neighborhood structure, search operators etc.) the performance of the heuristic depends *solely* on the fitness function being optimized. In this section we show that it is possible to represent the function as k independent parameters (the actual number depends on the algorithm). We refer to this decomposition of the function as the *information content* of the function. We suggest to use a greedy criterion to measure the performance and derive two measurements of problem difficulty.

Let X denote a finite search space and $f : X \to Y$ a function. Let F denote all possible fitness functions. Using Vose's notation [21], RSHs 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. 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. The search is a sequence of iterations of τ: $\Psi_k \xrightarrow{\tau} \Psi_l \xrightarrow{\tau} \cdots$.

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 *navigation* operator. The selection phase identifies solutions with high fitness value, the navigation operator samples accordingly new solutions from X.

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, s, which represents a possible way to choose (or select) solutions from Ψ_i. For example, in GAs, given a particular population, Ψ_i, s is a possible mating pool. For a (1+1) evolutionary strategy, s can be either the parent or the offspring. Given a state Ψ_i, we denote by S^i all possible ways of selecting points. That is, S^i is a set of multisets, each multiset, $s \in S^i$, corresponds to one possible way of selecting points.

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, s, that the algorithm selects, being the only argument for the navigation operator, *defines* (possibly, stochastically) 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 a fitness 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:

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

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:

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

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

$$P_f^i(s \mid \Psi_i, f) = \prod_{x \in d} \Pr(x \mid \Psi_i, f). \tag{3}$$

As previously mentioned, we focus only on the selection phase of the algorithm. Our analysis is done under the assumption that all the parameters of the algorithm (including the choice of representation or neighborhood structure) are defined. In this case, the performance of the algorithm depends on the fitness function alone, or using our formulation, the information content of the fitness function.

Since for each $f \in F$, $\mathcal{P}_f$ is properly defined, the set of all possible fitness functions F corresponds to a similar set, denoted by $\mathcal{P}_F$ of probability distributions. Let $\mathbf{P}(a, f)$ denote the performance of the algorithm, a, on the function f. We assume that $\mathbf{P}$ indicates the efficiency of the algorithm. It can, for example, denote the expected number of fitness evaluations required to find an optima point, but can be more general (e.g., expected number of generations for GA). We assume that the initial state is chosen uniformly at random and so, when the expectation is done over multiple runs, there is always some probability of the global optimum to be selected. Since we assume that a is fixed we consider the performance function as follows:

$$\mathbf{P} : \mathcal{P}_F \rightarrow \mathbb{R}$$

2.1 Greedy Criteria for Performance

Having no a priori assumption about $\mathcal{P}_f$ is equivalent to assuming that each $P_f^i \in \mathcal{P}_f$ is a uniform distribution over S^i. In that case, a distance between two states, $d(s, \Psi_j)$, can be defined as the expected first hitting time of a random walk starting from s and reaching Ψ_j.

Assuming that the algorithm tries at each step to minimize the distance to an optimum state a greedy criterion for the performance can be defined. Assuming that the optima is known, the efficiency of P_f^i can be measured as the expected distance at time step $t + 1$ to an optimum state – that is, $\sum P_f^i(s) d(s, \Psi_{opt})$ where Ψ_{opt} denotes a state which contains an optimum point. This suggests that the effect of each variable of the function $\mathbf{P}$ can be evaluated, to some extent, independently. Since, such an analysis can be done only if the global optimum is given, we write explicitly $\mathbf{P}(\mathcal{P}_f, x_{target})$ where $\mathcal{P}_f$ is the information content of the function f and x_{target} denotes the global optimum.

2.2 Distance and Performance

Following this line of reasoning, it is possible to define a distribution $\mathcal{P}_{f_{opt}}$ such that $d(s, \Psi_{opt})$ is minimized for each $P_{f_{opt}}^i$. The accuracy of any other distribution $\mathcal{P}_f$ can be evaluated by comparing how similar it is (e.g., using the expected Kullback-Leibler divergence) to $\mathcal{P}_{f_{opt}}$.

2.3 Number of Ties and Entropy

One thing that lead to poor performance in classical AI search is ties between competing alternative successor states [17]. This is because a searcher is then forced to either explores all of them or chooses one arbitrarily. The same happens in RSHs when a distribution P_f^i is uniform (or almost uniform). In that case, the algorithm chooses the next state uniformly at random. Clearly, if this is the case for all P_f^i's the algorithm performs random search.

A natural way of measuring the frequency of ties for stochastic algorithms is to consider the entropy of the distribution defined over all possible successors. Entropy measures how uniform a distribution is, that is, the larger the entropy the less informed the algorithm is.

Definition 2. *The entropy, $H(P_f^i)$, of the function f for the state Ψ_i, is defined as:*

$$H(P_f^i) = -\sum_{s \in S^i} P_f^i(s) \log P_f^i(s)$$

Depending on the function f, different search states, have different probabilities to occur in a run. For example, RSHs optimizing the Ridge function [11] are very likely to sample a state which contains the solution $x = 0^n$. So, we measure the overall effect of ties as the expected entropy:

$$E[H(f)] \equiv \sum_i Pr(\Psi_i) H(P_f^i)$$

where $\Pr(\Psi_i)$ denotes the probability that the state Ψ_i will be sampled during a run.

When $E[H(f)]$ is maximal, either the function is almost flat, or the selection mechanism is random. The needle-in-a-haystack (NIAH) is a well known example for this scenario: the fitness of all the solutions but the global optimum equals 0. The performance of RSH on the NIAH is bounded from below by $(|X|+1)/2$ [5] which suggests that high expected entropy implies hardness. On the other hand, when $E[H(f)] = 0$ nothing can be said about the performance (which depends, in this case, solely on the relation between the search operators and the fitness function). The effect of intermediate values of entropy on performance is more difficult to asses. Presumably, the higher the entropy the closer the performance to that of a random search. Under the first order approximation considered in the next section – this is precisely the case.

It is worth mentioning that NP-hard problems exist where the expected entropy is maximal. SAT is an obvious example: from the black box perspective, a SAT instance is a variant of the NIAH with possibly more than one needle. Interestingly, while the fitness distribution of many MAXSAT problems is not flat, a NIAH-type of MAXSAT problem can be generated for n bit space using, for example, the following formula:

$$\bigcap_{i \geq 0} (x_i \bigcup_{j<i} \neg x_j)$$

where $x_i \in X$ are literals. This is definitely not a typical case, however, this is a clear example for important problems for which the entropy, from the *black box perspective*, is maximal.

3 Approximation of the Performance Function

The decomposition of f to $\mathcal{P}_f$ is precise, however, due to its size, it cannot be used in practice. In this section we define a simpler decomposition, in which every function f is associated with a vector V. In the reminder of the section we relate the material in section 2 to this simpler model. In particular, section 3.1 introduces a first order approximation to the performance. Section 3.2 defines the distance between a vector V to an optimal vector V_{max} as a predictive measure of problem hardness. Finally, in section 3.3 the entropy of f is approximated by using V.

The size of $|\Psi|$ and $|\Psi_i|$ for realistic search algorithms is usually bounded. For example, genetic algorithms typically 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. As illustrated in section 2, for these algorithms, equation (1) (the probability, given two solutions, that one is selected) is the only one that considers, explicitly the fitness function. Equation (2) and then (3) (the actual distribution, P_f^i) depend only on $\Pr\limits_{\text{tmt}}$.

The codomain is the function $\Pr\limits_{\text{tmt}}$ is the set of probabilities $\{0, 0.5, 1\}$. Equivalently to equation (1), given a function f, we define the following indicator function

$$t(x_i \mid x_i, x_j) = \begin{cases} 1 & \text{if } f(x_i) > f(x_j), \\ 0 & \text{if } f(x_i) = f(x_j), \\ -1 & \text{otherwise.} \end{cases} \tag{4}$$

The codomain for t, $\{-1, 0, 1\}$, was chosen for the purpose of the approximation we use later in the section.

In our simplified model we define the information content of a function as a tuple (X, t) including a set X of configurations (the search space) and an indicator function $t : X \times X \to \{1, 0, -1\}$. For every pair (x_i, x_j) of elements in X, t indicates the preference (if any) of the algorithm for one of the solutions. Naturally, the function t can be represented as an $|X| \times |X|$ information matrix M with entries $m_{i,j} = t(x_i, x_j)$.

It is important to note that not all information matrixes can be associated to a fitness function (the information matrix not necessarily represents a partial order). We will call *invalid* those information vectors that cannot be derived from corresponding fitness landscapes.

Since $t(x_i, x_j) = -t(x_j, x_i)$, the matrix M presents symmetries with respect to the main diagonal which reduces the number of available degrees of freedom to the elements above (or below) the diagonal. So, in order to represent an information matrix more concisely, we use the following vector to store the relevant (above diagonal) entries in the information matrix:

$$V = (v_1, v_2, ..., v_n) = (m_{1,2}, m_{1,3}, ..., m_{|X|-1,|X|}), \tag{5}$$

where $n = (|X|^2 - |X|)/2$. Throughout the paper, we use the matrix notation only to illustrate, graphically, some concepts, otherwise we use the vector notation.

3.1 A First Order Approximation

The performance function $\mathbf{P}$, using our model, is a function of V and, as discussed in section 2, the target solution, x_{target}. That is, $\mathbf{P} : V \times X \rightarrow \mathbb{R}$. This may be a very complicated function. However, one might wonder whether a first order approximation

$$\mathbf{P}(V, x_{trgt}) \approx c_0 + \sum_{i=1}^{n} c_i v_i. \tag{6}$$

could be sufficient in order to model, to some extent, the performance of a simple GA.[1] We denote the vector $C = \{c_i\}$ as the *performance vector*. The approximation to the performance can be written in a vector notation as:

$$P(V, x_{trgt}) \approx c_0 + C \cdot V \tag{7}$$

In order to calculate the coefficients c_i we can apply statistical or machine learning techniques. For a fixed x_{target}, a training set is made up of pairs of the form $(V_k, \mathbf{P}_k)$, $k = 1, 2, \cdots$, where V_k is a particular information vector – an input for the learner – and $\mathbf{P}_k$ is the corresponding performance measure – the target output for the learner. Ideally, we would want $\mathbf{P}_k = E[\mathbf{P}(V_k)]$ (where the expectation is over multiple runs). Since we do not know the function $\mathbf{P}(V)$, we need to obtain the target values $\mathbf{P}_k$ by some other means. These values, for example, can be estimated by averaging the performance recorded by running the algorithm a suitably large number of times over the particular landscape in question. In order to estimate the coefficients c_i we apply multivariate linear regression over our training set.

Because of the dimensionality of C, this approach can only tackle small landscapes (e.g., 3 loci). In the following section, however, we develop an approach based on a notion of distance which allows us to apply our model for bigger landscapes (14 bits). In principle, an indication to the performance can be obtained also for realistic landscapes. However, in this case, one has to *sample* V and so the sampling noise is compounded with the errors already present due to the linearity of the approximation, resulting in unacceptable errors. As providing a new predictive measure of problem difficulty is not our main objectives, we do not explore scalability issues further in this paper.

[1] The purpose of approximating V rather than $\mathcal{P}_f$, is first and foremost, to be able to validate the first order approximation *empirically*.

3.2 A Predictive Measure of Problem Difficulty

In section 2.2 a distance between $\mathcal{P}_f$ and an optimal $P_{f_{opt}}$ was suggested as an indication to problem difficulty. However, this distance cannot be computed in practice. The model, V, makes it possible to calculate this distance explicitly. In this section we give an indication to problem difficulty which is based on a distance between an information vector, V, and an optimal information vector V_{max}. We conclude this section by arguing that instead of V_{max}, which is hard to compute, we can use the information vector of a known easy problem, making it possible to estimate the distance without calculating the performance vector C.

We begin by assuming that C is known. Therefore it is easy to construct an information vector $V_{\max} = (v_{\max_1}, \cdots, v_{\max_n})$ which contains only positive information. The performance of an algorithm on such a landscape is maximal:

$$v_{\max_i} = \arg\max_{v_i}[c_i v_i], \tag{8}$$

Optimal information vectors for a given set of coefficients c_i (algorithm) are those where $v_i = 1$ for all i where $c_i > 0$, $v_i = 0$ for all i for which $c_i < 0$, and v_i takes any value for all the remaining i's. The worst possible landscapes, $V_{\min}$ are constructed similarly (note $v_{\min_i} = -v_{\max_i}$ for all i where $c_i \neq 0$).

Given a landscape V, each entry v_i for which $v_i \neq v_{max_i}$, gives an indication to the expected difficulty of V. More generally, *the number of non-matching entries between V and $V_{\max}$ is a rough indicator of problem difficulty.* This is only an indicator because we do not consider the magnitude of the coefficients c_i, only their sign. We define the number of non-matching entries between two landscapes V_1, V_2 as the distance $d(V_1, V_2)$:

$$d(V_1, V_2) = \frac{1}{n}\sum_i |v_{1_i} - v_{2_i}|. \tag{9}$$

For landscapes without any 0 elements, the distance between two landscapes is the proportion of non-matching entries in the two vectors representing the landscapes.

The distance $d(V_{\max}, V)$ provides an indication to the difficulty of V. However, the set of coefficients C cannot be calculated for realistic problems, and hence $V_{\max}$ cannot be calculated. Instead, in the empirical validation we use an estimation of $V_{\max}$. This can be any landscape which is known to be very easy to optimize, V_{easy} (e.g., ONEMAX for GA). Once V_{easy} is given, the distance can be calculated and the hardness approximated. More formally, we propose to use as an *indicator of problem difficulty* the quantity

$$h(V) = d(V, V_{\text{easy}}), \tag{10}$$

where h is mnemonic for "hardness" and $d(\cdot,\cdot)$ is a distance measure between landscapes.

$h(V)$ gives an indication to problems hardness. The precision of this indicator depends, first of all, on the first order approximation assumption. Moreover, since it does not consider the magnitude of the coefficients c_i, it depends also on their distribution. Finally, since we use V_{easy} to approximate $V_{\max}$ it also depends on the distance, $d(V_{\max}, V_{\text{easy}})$, between the two. Despite these numerous approximations, as we will show in section 4.2 the approach produced very good results, suggesting that all approximations (including the pseudo-linearity of the performance function) are reasonable.

3.3 Indication for the Expected Entropy

In section 2.3 we argued that the number of ties for stochastic search algorithms can be measured as the expected entropy of the distribution P_f^i. However, this cannot be done in practice. Using the vector V we suggest to use as a replacement for the expected entropy the average number of pairwise ties, denoted $H(V)$, which can be easily calculated:

$$H(V) \equiv \frac{1}{n} \sum_i \delta(v_i = 0)$$

In the following we will still refer to $H(V)$ as the entropy of an information vector.

The first order approximation defined in the previous section suggests that the entropy gives an upper bound to the magnitude to which the algorithm can perform either better or worse than a random search. Performance is measured as $P(V) = c_0 + \sum v_i c_i$. $H(V)$ counts the number of entries with a value equal to 0. Clearly, the larger $H(V)$ the smaller the deviation $|P(V) - c_0|$ of the performance from random search can be.

$H(V)$ can be used in practice to calculate an indication to the entropy of the fitness function. This can be done by estimating the fitness distribution of the function, for which several methods exists (e.g., [20]). Once this is done, the number of solutions with equal fitness values can be estimated. We already gave in section 2.3 examples to NP-hard problems with maximum entropy. Intermediate values of entropy can help to tune the exploration–exploitation tradeoff of the algorithm: the higher $H(V)$ the more explorative (or randomized) the algorithm is. Presumably, the mutation rate for such functions should be smaller (and the other way around). We plan to investigate this in future research.

4 Empirical Evaluation

It is hard to assess mathematically the accuracy of our framework. This is because the framework is applicable to search algorithms in general, but, potentially, each algorithm has a different performance function $P(V)$. In addition, it is exceptionally hard to build an explicit formulation for this function, even when considering a specific search algorithm (e.g., a GA). So, empirical validation is the only viable strategy.

Focusing on GAs, in this section, we describe empirical evidence which strongly corroborates the framework. In particular, we show that the performance vector can be used in order to predict the performance of the algorithm on unseen problems. In Section 4.1, we conduct an exhaustive analysis on small landscapes (for the search space of binary strings of length 3) and show that this is indeed the case. Then, moving away from landscapes of a small size towards more realistic sizes (14 bits), in Section 4.2 we use various examples of known problems from the literature (e.g., multi modal landscapes, NIAH, MAXSAT, etc.) in order to show that the hardness of a problem can be estimated by measuring its distance from an easy reference landscape using Equation 10.

4.1 Exhaustive Analysis

In this section we test our main hypothesis and we show that our framework can be used in order to estimate the performance of a GA. Since this requires a full knowledge of the performance vector, we provide results for small landscapes.

We used a simple GA with one-point crossover applied with 100% probability. The takeover time (i.e., the time required for the entire population to converge to the target solution) was used as the performance measure (that is, in this case, we consider minimization). We used a population size of 14. The maximum number of generations was 500. The search on each landscape was repeated 1000 times so as to obtain accurate averages. The target solution (global optimum) was excluded from the first generation.

The experimental setup might look unusual. What is the purpose of using a population of size 14 and 500 generations to explore a search space of size 8? We decided to choose these settings in order to get higher resolution for the performance of a GA. For this purpose, the performance measure we chose is the takeover time (rather than, for example, the number of generations it takes to sample the optimum). The takeover time depends, once the optimum is sampled, on the selection pressure. However, firstly, we assume that the easier the landscape, the more copies of the optimum will be in early generations and so the faster the takeover time will be and secondly, since we use a very low selection pressure (tournament of size 2), the influence of a random occurrence of an optimum will not be crucial.

In a first experiment, we measured the mean takeover time for *all possible valid landscapes*. These are landscapes which can be derived from a fitness function and where none of the elements is 0. It is important to emphasize that the target solution was fixed. Therefore, we were measuring the performance of a GA on all possible landscapes given that the optimum is at a particular position in the search space. Since the size of the search space X is 8, the reduced search space X' is of size 7 and, so, we have $7! = 5040$ possible landscapes. In order to estimate the performance vector (Equation 6) we used multivariate linear regression on the results obtained from running the GA over all such landscapes. The correlation coefficient between observed and predicted performance is 0.935.

In order to verify whether the linear approximation to $P(V)$ generalizes well, we sampled 1000 additional landscapes out of the entire space of possible information vectors (i.e., including invalid landscapes, see Section 3). The correlation between prediction and observation is still very high (0.923), suggesting good generalization. For more details about the multivariate linear regression the coefficients obtained and a possible interpretation see [3].

4.2 Estimation of Problem Hardness

In the previous section we have shown that our framework can be used in order to accurately estimate the performance of a GA and assess problem difficulty. However, it is clear that a direct estimation of the performance vector can only be used for small search spaces. In this section, rather than using the performance vector directly, we use the ideas presented in Section 3.2 to see if the approach can be applied to more practical scenarios.

In Section 3.2 we argued that the hardness of a problem can be estimated using the distance of its information vector from the optimal landscape $V_{\max}$. Since, in general, the optimal landscape is not known, we proposed to use an approximation, V_{easy}, instead (Equation 6). The question now is which easy problem to choose. We know from many empirical studies that unimodal problems tend to be GA-easy, the Onemax problem being a glowing example. The Onemax belongs to the following more general class of functions: $f(x) = \sum_i \delta(x_i = x_{\text{target}_i})$ where x_{target} is the global optimum. Onemax, is a specific case, where x_{target} is the string of all ones. In this work we decided to use the information vector V_{easy} derived from $f(x)$ as an approximation of the optimal landscape $V_{\max}$.

The information vector and the performance function are defined for a fixed target solution. If we change the target solution the same information vector can change from being easy to being difficult (e.g., consider the information vector induced by the Onemax function where we change the optimum to being the string 000). So, the distance between landscapes must be computed for landscapes with the same global optimum. This requires knowing the global optimum in advance.

In the following experiments, we calculated $h(V)$ as the distance between the actual landscape induced by a problem and the one induced by the function $f(x)$ using the global optimum of the problem as x_{target}. We used a simple GA with uniform crossover applied with 100% probability and mutation applied with 10% probability. The search space included binary strings of 14 bits. The population size is 20. The first generation in which the optimum was found was used as the performance measure. The results are averages of 100 runs.

The remainder of this section is organised as follows. First we give empirical results for various problems then we test our approach on three counterexamples for other measures of problem difficulty.

Hardness of standard test problems. In this subsection we estimate the hardness of problems with no information (NIAH), random information or random problems (RAND), maximally reliable information (unimodal functions) and maximally unreliable information (deceptive functions). Furthermore, we study problems with a variable level of difficulty: the NK landscapes with k =1–10, multimodal landscapes (MM1, MM2, etc.) with a varying number of local maxima (1–20), and trap functions ($TRAP_i$ where $i \in \{1, \cdots, l\}$ indicates the level of difficulty, $TRAP_1$ being the most difficult). Finally, to test our measure of difficulty on landscapes which were not induced by artificial problems, we also considered 12 random MAXSAT problems (14 literals, 59 clauses). For each problem, we consider only one global optimum. If a problem had more than one global optimum, we chose one at random to be the target solution.

Table 1 gives our (predicted) measure of difficulty ($h(V)$) for selected problems and the actual performance obtained by the GA (P). Note that $h(V)$ is scaled between 0 (very easy) and 1 (very hard) while the performance P between 1 (when, on average, the optimum was sampled in the first generation) and 100 (on average, the optimum was not sampled in 100 generations).

As one can observe, the scale of $h(v)$ is neither linear nor always consistent. For example, a difference of only 0.002 between MM5 (0.403) and NK2 (0.405) corresponds to a very large difference in performance: 68.7 for MM5 vs. 33.9 for NK2. Still, the correlation coefficient between observed and predicted difficulty (for all problems) is 0.82.

The table confirms some of the previous results regarding GA hardness (see section 1). For example, the table shows that multimodality is not a good indicator of problem difficulty [11]. A landscape with 9 local maxima has the same expected difficulty as a landscape with 16 local maxima, while a landscape with 5 local maxima is more difficult than a landscape with 16.

Table 1. Estimated hardness $h(V)$ and average performance P for a selection of 40 test problems

$h(V)$	P	Problem	$h(V)$	P	Problem	$h(V)$	P	Problem	$h(V)$	P	Problem
0.000	11.4	MM1	0.363	77.3	MM3	0.405	71.4	MM12	0.493	87.9	RAND
0.226	40.1	MM2	0.372	52.8	MM6	0.422	90.5	NIAH	0.496	83.6	NK9
0.293	23.0	NK1	0.374	59.3	MAXSAT	0.427	52.9	NK4	0.497	88.0	NK6
0.306	27.5	MAXSAT	0.374	58.8	MM9	0.44	74.8	MM13	0.502	89.2	RAND
0.325	28.2	MAXSAT	0.384	60.4	MM16	0.452	70.1	MM7	0.507	90.7	MM14
0.325	48.7	MAXSAT	0.387	53.7	MAXSAT	0.463	79.5	MM17	0.509	89.0	NK7
0.334	56.9	MM4	0.393	55.7	MM11	0.463	65.5	ALTNBR	0.511	87.0	NK8
0.336	39.3	MAXSAT	0.394	51.1	MAXSAT	0.473	82.9	MM15	0.605	79.3	TRAP4
0.355	42.6	MAXSAT	0.403	68.7	MM5	0.473	76.01	NK3	0.756	99.0	TRAP3
0.361	51.4	MAXSAT	0.405	33.9	NK2	0.491	90.4	MM18	0.842	100	TRAP1

The table also confirms that the NK model is not appropriate for the study of problem difficulty because problems with a $k > 2$ are already very difficult [11]. Indeed, our measure suggests that the difficulty of such landscapes is close to random.

Different instances of the same problem might have different degrees of difficulty in the black-box scenario [11]. Indeed, the predicted difficulty for different instances of the MAXSAT problems varies from 0.306 (easy) to 0.394 (difficult) – even though, all were chosen with the same variable to clause ratio.

Hardness of known counterexamples for other difficulty measures. In the previous section we presented evidence supporting the hypothesis that $h(V)$ is a meaningful indicator of problem difficulty. In this section we test our framework on three problems where other measures of difficulty have been shown to fail.

Naudts and Kallel [16] constructed a simple problem consisting of a deceptive mixture of Onemax and Zeromax, where both the fitness distance correlation (FDC) measure and the sitewise optimisation measure (a generalisation for the FDC and epistasis suggested in the same paper) failed to correctly predict performance. For this class of problems, the higher the mixture coefficient m, the harder the problem, yet no hardness measures was able to predict this. We performed experiments with this problem[2], with the control parameter m varying from 1 (easy) to 9 (hard). The correlation between the predicted difficulty and the actual performance was 0.75. So, we were largely able to capture the difference in performance as the parameter m varied.

Jansen [11] showed that the fitness distance correlation of the ridge function is very small. Yet, this is an easy problem for a hill climber. So, we decided to apply our method to this function as well. The distance of the ridge function to the optimal landscape is 0.84, which indicates a very difficult problem. Indeed, the GA was not able to find the solution in 100 generations. A problem that is easy for a hill climber is not necessarily easy for a recombinative GA.

Jansen [11] gave two counter examples to the bit-wise epistasis measure of difficulty. The first one was the NIAH which we already discussed before. The second was the leading-one function. The distance of the leading one function from the optimum landscape is 0.36, which predicts well the performance shown by a GA (50.3 average number of generations required to sample the global optimum).

5 Conclusions

The decomposition of f into $\mathcal{P}_f$ captures the way in which the algorithm uses the fitness function – i.e., via the selection paradigm – to define a probability distribution over sampled points which is used to pick the solutions around which to expand the search.

This decomposition enables one, to a large extent, to approximate the performance of a GA using a simple, linear function. In section 4.1 we demonstrated this empirically for problems of small size (3 bits). Also, a predictive measure based on the linear approximation was used to assess the GA hardness (section 4.2) of several larger problems (14 bits).

Ties in classical graph search algorithms are known to lead to poor performance. We assessed the influence of ties for RSHs using the expected entropy of $\mathcal{P}_f$. Interestingly, the linear approximation predicts that the number of ties as measured in section 3.3 gives a bound to the expected performance. In the extreme case where the entropy is maximal performance cannot be better than that of random search.

[2] The GA used in [16] is different from the one used here.

While the framework presented in section 2 is general, the first order approximation was tested only for GAs. We expect the same approximation to hold for other population based, stochastic algorithms. However, this remains to be checked.

The drawbacks of this framework is the lack of bounds to its precisions. The approximation suggested in this paper and the conclusions which can be derived from it can be used only as a first indication to the properties of particular algorithms or functions. Nevertheless, the ever increasing number of new complex metaheuristics makes a rough, but quick estimation a necessity.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable suggestions, which have improved the paper greatly.

References

1. Altenberg, L.: Evolving better representations through selective genome growth. In: Proceedings of the 1st IEEE Conference on Evolutionary Computation, Orlando, Florida, USA, June 27-29, 1994, vol. 1, pp. 182–187. IEEE, New York (1994)
2. Altenberg, L.: NK fitness landscapes. In: Handbook of Evolutionary Computation, pp. B2.7.2. Oxford University Press, Oxford (1997)
3. Borenstein, Y., Poli, R.: Information landscapes and the analysis of search algorithms. In: GECCO '05. Proceedings of the 2005 conference on Genetic and evolutionary computation, New York, NY, USA, pp. 1287–1294. ACM Press, New York (2005)
4. Davidor, Y.: Epistasis variance: A viewpoint on GA-hardness. In: Rawlins, G.J.E. (ed.) Proceedings of the First Workshop on Foundations of Genetic Algorithms, Bloomington Campus, Indiana, USA, July 15-18, 1990, pp. 23–35. Morgan Kaufmann, San Francisco (1990)
5. Droste, S., Jansen, T., Wegener, I.: Upper and lower bounds for randomized search heuristics in black-box optimization. Electronic Colloquium on Computational Complexity (ECCC) (048) (2003)
6. Forrest, S., Mitchell, M.: Relative building-block fitness and the building block hypothesis. In: Whitley, L.D. (ed.) Proceedings of the Second Workshop on Foundations of Genetic Algorithms, Vail, Colorado, USA, July 26-29, 1992, pp. 109–126. Morgan Kaufmann, San Francisco (1992)
7. Rothlauf, F.: Representations for Genetic and Evolutionary Algorithms. Springer, Heidelberg (2002)
8. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, London (1989)
9. Goldberg, D.E.: Making genetic algorithm fly: a lesson from the wright brothers. Advanced Technology For Developers 2, 1–8 (1993)
10. Grefenstette, J.J.: Deception considered harmful. In: Whitley, L.D. (ed.) Proceedings of the Second Workshop on Foundations of Genetic Algorithms, Vail, Colorado, USA, July 26-29, 1992, pp. 75–91. Morgan Kaufmann, San Francisco (1992)
11. Jansen, T.: On classifications of fitness functions. In: Theoretical aspects of evolutionary computing, pp. 371–385. Springer, London, UK (2001)

12. Jones, T., Forrest, S.: Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: Proceedings of the 6th International Conference on Genetic Algorithms, San Francisco, CA, USA, 1995, pp. 184–192. Morgan Kaufmann Publishers Inc. San Francisco (1995)
13. Kauffman, S.: The Origins of Order: Self-Organization and Selection in Evolution. Oxford University Press, Oxford (1993)
14. Moraglio, A., Poli, R.: Topological interpretation of crossover. In: Deb, K., et al. (eds.) GECCO 2004. LNCS, vol. 3102, pp. 1377–1388. Springer, Heidelberg (2004)
15. Naudts, B.: Measuring GA-hardness. PhD thesis, University of Antwerpen, Antwerpen, Netherlands (1998)
16. Naudts, B., Kallel, L.: A comparison of predictive measures of problem difficulty in evolutionary algorithms. IEEE Trans. Evolutionary Computation 4(1), 1–15 (2000)
17. Pearl, J.: Heuristics: intelligent search strategies for computer problem solving. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA (1984)
18. Radcliffe, N.J.: Equivalence class analysis of genetic algorithms. Complex Systems 5, 183–205 (1991)
19. Rana, S.: Examining the Role of Local Optima and Schema Processing in Genetic Search. PhD thesis, Colorado State University, Colorado, U.S.A (1998)
20. Rose, H., Ebeling, W., Asselmeyer, T.: The density of states - a measure of the difficulty of optimisation problems. In: Parallel Problem Solving from Nature, pp. 208–217 (1996)
21. Vose, M.D.: The Simple Genetic Algorithm: Foundations and Theory. MIT Press, Cambridge, MA, USA (1998)