

# Information Landscapes and the Analysis of Search Algorithms

Borenstein Yossi  
University of Essex  
yboren@essex.ac.uk

Riccardo Poli  
University of Essex  
rpoli@essex.ac.uk

## ABSTRACT

In [15] we introduced the *information landscape* as a new concept of a landscape. We showed that for a landscape of a small size, information landscape theory can be used to predict the performance of a GA without running the algorithm. Based on this framework, here we develop a new theoretical model to study search algorithms in general. Particularly, we are able to infer important properties of a search algorithm without having knowledge about its specific operators. We give an example of this technique for a simple GA.

## Categories and Subject Descriptors

F.2.0 [Theory of Computation]: Analysis of algorithms and problem complexity.

## General Terms

Algorithms, Performance, Theory.

## Keywords

Fitness landscape, Genetic Algorithm, Theory

## 1. INTRODUCTION

During the last 20 years many algorithms (metaheuristics) have been proposed in order to explore black-box problems [1]. Usually a search algorithm tries to infer the position of good new solutions in the search space based on previously sampled solutions.

Many metaheuristics have been applied successfully to an ever increasing number of hard combinatorial optimization problems such as TSP, vehicle routing, job shop scheduling, and bin packing. However, in many cases, their remarkable empirical success is not associated to corresponding robust theoretical foundations.

Fundamentally, the reason for this is that the intrinsic complexity of modern metaheuristics makes it difficult to explore their dynamics theoretically. Most of them combine more than one search operator. Since it is difficult to analyze the effect of even a single operator, clearly, the interaction between multiple operators

makes the analysis even more difficult.

The analysis of a search algorithm usually follows one of the following approaches. A first one tries, despite the difficulties mentioned above, to give a probabilistic analysis which accounts for the effect of all the operators in the algorithm. A second focuses on specific operators. A third approach tries to infer properties which might make a problem either difficult or easy for the algorithm to search.

In any case, it is usually easier to construct a theory for restricted scenarios, e.g. for problems with specific properties. Therefore, an exact analysis is often given only for specific artificial problems.

We exemplify this, focusing, for the sake of brevity, on the Genetic Algorithm (GA). Since this is one of the most popular metaheuristics we think this is an interesting case-study.

The main tools that have frequently been used in the literature to study search algorithms are either very easy landscapes or very difficult ones.

Easy landscapes provide an intuition as for the scenarios in which the algorithm performs well. Usually, it is easier to construct a theory restricted to those scenarios and validate it with empirical results. The royal road function is an example [2] of this approach although the attempt to create easy landscapes failed. The extensive investigation of problems like the onemax [3] is another. Difficult landscapes provide similar intuition for situations where an algorithm fails.

The use of easy and difficult problems is widespread. However, the definition of “easy” or “difficult” is problematic. A landscape can be easy or difficult only w.r.t a particular reference. When considering a new algorithm reference performances are not available. They need to be discovered through an extensive theoretical and empirical investigation.

Furthermore, the definition of difficult problems is fuzzy. The needle-in-a-haystack is a well studied difficult problem [4]. However, it is quite clear that it is difficult in a different way from, for example, a fully deceptive problem [5]. Even though, intuitively, the difference between the two is obvious, there is no explicit definition to distinguish between them.

The simple GA uses a finite population. Its main operators are crossover and mutation. It can be applied to problems with different representations (neighborhood structures). Given the difficulty of analyzing the combined effect of all its operators and the different possible neighborhood structures, many researchers study the different operators separately.

In [6][7] some interesting results are obtained for a mutational-based GA. The properties of crossover are studied in [8], however only for the artificial problem, onemax. The convergence

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

GECCO'05, June 25–29, 2005, Washington, DC, USA.  
Copyright 2005 ACM 1-58113-000-0/00/0004...\$5.00.

properties are studied under the influence of selection only (drift) in [9].

The dynamical system approach [10] and schema theory [11] are attempts to study the combined effect of all the operators for a simple GA. These are successful but the development of such models for new search operators and neighborhood structures takes a lot of time and effort. This is a problem especially when considering the constant emergence of new variants of the simple GA. Among them are the variable length GA [12], new biologically inspired algorithms [13], redundant representations [14] and many others. Theoretical approaches are not likely to keep up with all the new variants.

In this paper we use the *information landscape* framework we introduced in [15] to study search algorithms. The first section gives a background on the framework. In section 3, a robust definition of easy and difficult problems is given and we show how the hardest and easiest problems for an algorithm can be constructed *without* using any knowledge of the algorithm. An example is given for a simple GA. Section 4 provides a way to assess the combined effect of the different search operators and the neighborhood structure. In section 5 we consider this explicitly for a local search algorithm. We conclude with a discussion and conclusions (section 6).

## 2. Background

In [15] we proposed a redefinition of the concept of landscape that makes the quantity and quality of the information available to guide a search algorithm explicit. This is why the new landscape was called an *information landscape*.

The performance of any search algorithm on any particular information landscape can be approximated. In order to do so, we introduced the notion of *performance landscape*, which was then used to predict the performance of a GA over landscapes of a very small size (all 3-bit problems).

Since the work in [15] is the starting point for this paper, in the next sections we define the notions of information and performance landscape and discuss interpretations of the two concepts.

### 2.1 Information Landscapes

An *information landscape* is a triple  $(X, \mathcal{X}, t)$  including: 1) a set of configurations  $X$ , 2) a notion  $\mathcal{X}$  of neighborhood, nearness, distance or accessibility on  $X$ , and 3) a stochastic information function  $t: X \times X \rightarrow [0, 1]$ .

For every pair  $(x_i, x_j)$  of elements in  $X$ ,  $t$  gives the probability that  $x_i$  is superior to  $x_j$ . The value of the function  $t$  can be viewed as the outcome of a stochastic tournament selection with tournament size two. Naturally, the function  $t$  can be represented as an  $|X| \times |X|$  information matrix  $M$  with entries  $m_{i,j} = t(x_i, x_j)$ .

Note that when  $X$  is implied we can use the term *information landscape* to denote  $M$  without ambiguity.

The notion of information landscape does not require the availability of a fitness function. However, when a fitness function  $f$  is available, we should normally assume:

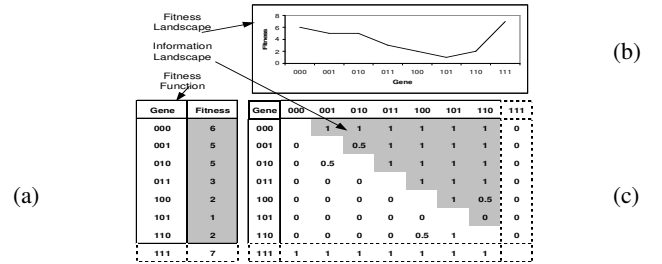
$$t(x_i, x_j) = \begin{cases} 1 & \text{if } f(x_i) > f(x_j) \\ 0.5 & \text{if } f(x_i) = f(x_j) \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

If the fitness function is noisy,  $t$  can take values other than 0, 0.5 and 1. Given the information landscape we can construct the following rank-based fitness function:

$$f_{\text{rank}}(k) = \sum_j m_{k,j} \quad (2)$$

Note that not all information landscapes can be associated to a fitness function (the information matrix may not induce a partial order). We will call *invalid* those information landscapes that cannot be derived from a corresponding fitness landscape.

Figure 1 gives an example of a fitness function, a landscape defined over a real neighborhood structure and the matrix which represents our *information landscape* for a bit-string configuration space.



**Figure 1. Three ways of representing the information given to a search algorithm: a) a fitness function (represented as a vector) b) a graph, representing topological properties (fitness landscape) and c) a matrix representing the outcome of all possible comparisons (information landscape).**

Since  $t(x_i, x_j) = 1 - t(x_j, x_i)$  the matrix (figure 1) presents symmetries with respect to the diagonal; the gray area marks the independent elements of the information landscape. Diagonal elements (omitted for clarity) are all 0.5. Moreover, we exclude the entries related to the optimum. We assume that we have a way to identify it, hence once it is found, the search is over.

In order to account for all this in a simple way we use a vector to store the relevant entries in the matrix:

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

where  $|V| \equiv n = (|X| - 1)(|X| - 2) / 2$ .

This definition of a landscape allows us to easily define the distance between two landscapes. Let  $V_a, V_b$  be two information landscapes, the distance between them is defined as:

$$d(V_a, V_b) = \frac{1}{n} \sum |v_{a_i} - v_{b_i}| \quad (3)$$

In addition we are in a position to *quantify* the amount of information present in a landscape. The *degree*  $d^{0.5}$  of the *information landscape* is the degree to which the information in the matrix available to an algorithm is different from 0.5. Formally, it is the distance between a landscape and the landscape where all matrix elements are 0.5 normalized to the range [0, 1]:

$$d^{0.5}(V) = \frac{2}{n} \sum |v_i - 0.5| \quad (4)$$

## 2.2 Performance Landscapes

Let  $P: V \rightarrow \mathfrak{R}$  be a performance measure over the landscape. For example,  $P$  could be the number of fitness evaluations required to find the global optimum.

$P$  is a complicated function of  $n$  variables for which we have no explicit formulation. However, this function can be estimated using machine learning techniques.<sup>1</sup> As an approximation for  $P$ , in [15] we adopted an  $n$ -variate linear function of the form

$$P(V) \cong c_0 + \sum c_i (v_i - 0.5) \quad (5)$$

and we used multivariate linear regression to estimate the coefficients. We then defined the array  $C = (c_i)$  as the *performance landscape*.

In [15] we indicated how, for a given performance landscape  $C$  and a degree of information  $d_0$ , we should expect our algorithm to provide best performance on the following information landscape:

$$V_{\max} = \left( \arg \max_{v_i} [c_i (v_i - 0.5)] \mid d^{0.5}(V_{\max}) = d_0 \right) \quad (6)$$

## 2.3 Interpretation

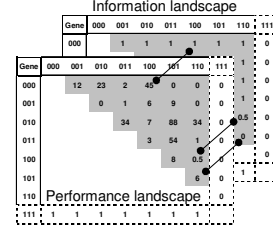
It is important to understand how an entry  $v_i$  in the information landscape and the corresponding coefficient  $c_i$  in the performance landscape are related to the performance of an algorithm. (See Fig. 2.)

The assumption underlying most optimization algorithms is that applying the search operators to solutions with high fitness (as opposed to ones with low fitness) is more likely to yield solutions close to the optimum (we only consider optimizations problems). In [15], we termed the chance of finding the optimum by applying the search operators on a point as the *effective distance* of that point from the optimum. For example, in the case of a GA, given a particular population, the effective distance of a string from the optimum would be the probability that given that this string is selected into the mating pool, the optimum will be found during the run. The effective distance is only a function of the search operators and the neighborhood structure.

The fitness of a solution is not related to the effective distance of the solution from the optimum. However, the algorithm uses the fitness of the solution as an indicator for such a distance. The performance of the algorithm depends on the correlation between the *relative fitness* (i.e. the *information* given by the fitness function) and the effective distance from the optimum

An entry in the *information landscape* represents therefore the *assumed* relative effective closeness to the optimum (i.e. if  $m_{i,j}=1$  solution “i” is closer to the optimum than solution “j”). Each element of the *performance landscape*, on the other hand, represents the degree to which the effective distance of one solution is closer to the optimum than another. In other words, the information landscape states *which* solution should be assumed to be better whereas the performance landscape states *whether*

*indeed and by how much* a solution is better than another for the purpose of eventually solving a problem.



**Figure 2. Relation between the information landscape and the performance landscape. High values in the performance landscape indicate that the corresponding entries in the information landscape are important.**

## 2.4 The structure of the paper

The objective of this paper is to show how the performance landscape can be used, directly and indirectly, in order to analyze new algorithms.

There is no need for explicit knowledge about the algorithm. As explained in the previous section, computing the *performance landscape* is a simple, 100% empirical procedure.

Once the size of the landscape is chosen (due to the computational cost, it cannot be a big landscape) a sufficiently large set of pairs  $(V, P)$  can be used as a training set to estimate  $P$ . We use a multivariate linear regression for this purpose. Once the performance landscape is known, the methods suggested in this paper can be applied.

Each method is first explained and then a concrete example using a GA is given. The results obtained for the GA are validated using schema analysis.

In all the examples the performance landscape is constructed using two versions of a simple GA. In the first one-point crossover is used. In the second, uniform crossover is used. The crossover is used with 100% probability. The takeover time (i.e. the time it takes to the entire population to converge to the target solution) is used as the performance measure. We use a population size of 12. The maximum number of generations is 400. The search on each landscape was repeated 100 times. The results are the average of those runs. The target solution (global optimum), which, without loss of generality, was the string “111”, is excluded from the first generation.

We measured the mean takeover time for a sample of 100 valid landscapes of degree 1 (full information). In order to estimate the performance landscape (equation 5) we did regression on the results obtained from running the GA over all such landscapes.

In section 3 we show how we can use the performance landscape indirectly in order to analyze an algorithm. In section 4 we show how to do it directly.

## 3. Indirect measures to explore an algorithm

In this section we show how the performance landscape can be used indirectly in order to facilitate the analysis of an algorithm. In particular, we show that this framework can be used in order to construct case-studies. We focus on the hardest problem and the easiest one. Rather than *using* them in order to analyze a GA, we show that both problems coincide with similar, well studied

<sup>1</sup> The training set includes examples of the form  $(V, P)$ ,  $V$  being an information landscape and  $P$  being an estimate of  $P(V)$  obtained by running an algorithm on  $V$  and measuring performance.

problems, in the literature (i.e. onemax and deceptive problem). The main idea is to demonstrate how beneficial case studies can be constructed automatically, without using any explicit knowledge about the algorithm.

In the next subsection we show how different case-studies can be constructed automatically. In the following one we give a robust definition to hardness.

### 3.1 Automatic creation of case studies

The performance landscape can be used in order to predict the expected performance of the algorithm for *any* information landscape. It represents the coefficients (equation 5) that are used in order to calculate the performance. Each element in the performance landscape relates to a specific element in the information landscape (figure 2). The coefficients  $c_i$  can be either positive or negative. The corresponding value for  $(v_i-0.5)$  in equation 5 can be either positive or negative as well.

The information landscape on which the algorithm is expected to have the best performance is, therefore, the one which is completely aligned to the coefficients of the performance landscape. That is, negative entries in the performance landscape should correspond to entries with a value of 0 in the information landscape (similarly for  $c_{i>0}$ ,  $v_i$  should be 1). This gives a positive contribution to the performance. See  $V_{\max}$  in equation 6.

Following similar reasoning the worst (hardest) landscape can be constructed. Moreover problems with any degree of difficulty can easily be created using this framework. It can be done in two ways. The first is to explicitly use the performance landscape. The second is simply to construct any landscape which is a combination of the best landscape and a random one.

In section 3.1.1, we demonstrate the creation of a landscape which gives an optimal performance. In section 3.1.2 we create a landscape which gives the worst performance. We show using a static schema analysis that these coincide with the classic unimodal and fully deceptive problems. The only difference is that our landscapes were constructed in an automated way without having any knowledge about the GA.

#### 3.1.1 The optimal landscape

We used equation 6 in order to construct the optimal information landscapes for the algorithms. Using equation 2 we constructed a corresponding ranked based fitness functions<sup>2</sup>.

**Table 1. Static schema analysis of the optimal landscape as predicted by the performance landscape for a simple GA with one point crossover.**

String	Fitness	Order	3	2	1	0				
000	0	Schema	111	11*	*11	**1	*1*	1**	***	
010	1	Fitness	7	6.5	5.5	6	4.75	4.75	4.75	3.5
100	2	Schema		01*	*01	*01	**0	*0*	0**	
001	3	Fitness		3	4	3.5	2.25	2.25	2.25	
101	4	Schema		10*	*10	*10				
011	5	Fitness		3	4	3.5				
110	6	Schema		00*	*00	*00				
111	7	Fitness		1.5	0.5	1				

<sup>2</sup> This is only one of many possible fitness landscapes. However, since we explicitly consider tournament selection, we don't need to consider all other equivalent landscapes.

Table 1 gives a static schema analysis for the best predicted landscape for one point crossover. Schema analysis is known to be the right tool to analyze the search conducted by the crossover operator. According to the building blocks hypothesis, an easy problem would be such that any schema that contains the optimum (in our case, the string 111) has higher fitness than its "competitors". The table reveals that this is indeed the case. Thus, without having knowledge about the algorithm, the prediction made using the performance landscape coincides with that of the schema theorem [16].

The landscape which was predicted for uniform crossover was not a valid landscape. Therefore, we were not able to do a schema analysis for it.

These two examples considered a landscape with complete information (a degree of information equals to 1). However, in reality this is not always the case. Therefore we constructed the optimal landscape for a smaller degree of information. In particular, we wanted to check whether the optimal landscape as predicted by our model coincides with other known GA-easy landscapes. The onemax problem is probably the most studied problem of such a kind. We chose, therefore, to find the optimal landscape for the degree 15/21 (the degree of onemax). The same results were obtained for both the one point crossover performance landscape and the uniform crossover one. The predicted optimal landscape was indeed onemax.

#### 3.1.2 The worst landscape

Using dual of equation 6 we constructed the worst landscapes as well. Table 2 gives a static schema analysis for the worst predicted landscape for one point crossover. The fitness of each string was calculated according to its rank (equation 2). The table reveals that the average fitness of each schema that contains the global optimum (111) is smaller than anyone of its competitors. Thus, it actually describes a fully deceptive landscape.

**Table 2. Static schema analysis of the most difficult landscape as predicted by the performance landscape for a simple GA with one point crossover.**

String	Fitness	Order	3	2	1	0				
000	0	Schema	111	11*	*11	**1	*1*	1**	***	
010	1	Fitness	7	3.5	4.5	4	3.25	3.25	3.25	3.5
100	2	Schema		01*	*01	*01	**0	*0*	0**	
001	3	Fitness		3	2	2.5	3.75	3.75	3.75	
101	4	Schema		10*	*10	*10				
011	5	Fitness		3	2	2.5				
110	6	Schema		00*	*00	*00				
111	7	Fitness		4.5	5.5	5				

### 3.2 Assessing hardness

As stated in the introduction the definition of easy or difficult problems is not simple. Particularly in the EC field it is not clear what an easy problem is and what a difficult one is.

In this section we provide a robust definition of difficulty of problems w.r.t any algorithm. We do so by giving three reference points: the easiest, the hardest and a random problem. The performance on any other problem can be assessed based on these.

The construction of the easiest and the most difficult landscapes was explained in the previous section. In this one we focus on the definition of a random search. It might seem trivial but as we will explain, it is not.

There are three possible scenarios:

1. The algorithm searches explicitly in a random way (random search).
2. The algorithm does not search in a random way, but there is no information in the landscape to guide the search.
3. The algorithm does not search in a random way but the information given by the landscape is random.

With respect to our framework the three ways are equivalent to: 1) a *performance* landscape with all entries equal to zero (the expected performance on any possible problem, in the case of optimization, is the same) 2) an *information* landscapes with all entries equal to 0.5 (degree of information equals to zero) and 3) an *information* landscape with no correlation with the performance landscape (see equation 5).

Randomness is usually a property of a difficult problem, but the three different types of randomness describe in practice three different degrees of difficulty.

The first algorithm, random search, does not assume anything about the structure of the landscape and hence its performance depends only on the size of the landscape (only optimization problems are considered).

In the second case, i.e. when there is no information in the landscape, one might think that the performance of an algorithm should be equivalent to that of random search. This is not the case. The search operators of the algorithm induce a bias on the search. This bias usually makes the algorithm less efficient than a random search [17].

Generally, based on equation 6, a random landscape and a landscape with no information should give the same performance. The difference between these two landscapes is that for the first the outcome of every tournament (an entry in the information landscape) is randomly fixed, whereas for the second it is random. The extent to which the performance measured on these two landscapes differs gives an indication of the non-linearities in the performance measure  $P$ .

It is worth noting that these three scenarios and their corresponding performances can be used in order to characterize the algorithm. E.g. the extent to which random search performs better on a random landscape than the algorithm might be an interesting measure of the sensitivity of the algorithm to noise.

Since in this paper we focus on assessing difficulty of problems for a given algorithm, we will select the performance of the algorithm on a random landscape as our reference (threshold) to divide easy from hard problems.

## 4. Direct measures to explore an algorithm

In the previous section we were able to demonstrate how the performance landscape can be used in order to construct different case studies. These can be used to indirectly infer the properties of the algorithm. We showed, using schema analysis, that the easiest and most difficult landscapes coincide with a unimodal landscape

and a fully deceptive problem. Moreover, we suggested using the performance on random landscapes as a reference point to the performance for the algorithm.

In this section we demonstrate how to infer properties of the fitness landscape. This can be done by exploring the *performance landscape*. In section 4.1 we explain how the performance landscape can be interpreted. In section 4.2, we show how this can be done in practice. We conclude (section 4.3) with a partial validation of the results obtained in the examples given in sections 4.1 & 4.2.

### 4.1 Understanding the performance landscape

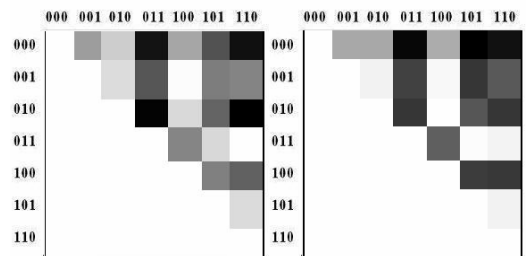
Since the analysis suggested in this paper is mainly dependent on the performance landscape, we give, in this section, additional explanations of its components.

We start with an example. Consider a (bit-flip) hill climber on a unimodal landscape. Let us assume that tournament selection is used in order to pick the initial starting point of the search. In the case of a binary representation the closer the initial search point to the optimum (w.r.t the neighborhood structure) is, the better the performance of the algorithm will be. Let 111 be the optimum. The optimal information landscape can tell us that the string 011 is a better candidate than 001, and that 011 is better than 000 as well. However, the extent to which 011 is closer (in the bit-flip sense) to the optimum when compared with 000 is higher than for 001. Therefore, the magnitude of element  $c_{011,000}$  in the performance landscape is expected to be bigger than  $c_{011,001}$ .

In general, given the choice between two alternative solutions as the basis for the next step of the search, ideally, the algorithm should choose the one that will maximize the probability to find the optimum and minimize the number of the required steps. Each entry in the performance landscape can be interpreted, therefore, as an *indicator* of the difference in the effective distance (or the conditional probability of the algorithm to find the optimum) given that either of the two points is selected. That is:

$$c_{i,j} \cong \alpha(p(X_{target} | X_i \wedge \neg X_j) - p(X_{target} | X_j \wedge \neg X_i)) \quad (7)$$

where  $p(X_{target} | X_i \wedge \neg X_j)$  is the probability that the algorithm will find the target solution given that  $X_i$  is chosen in the tournament between  $X_i$  and  $X_j$  and  $\alpha$  is a constant.



**Figure 3. A gray-scale image representing the relative values of the performance landscape. On the left is the performance landscape of one point crossover, on the right is the performance landscape of uniform crossover.**

Figure 3 shows (as grey scale images) the two performance landscapes that were introduced in section 3.1. A dark color represents a relevant entry (a coefficient with high absolute value) and a light color an insignificant one.

A quick look in the table shows us, for example, that for the two landscapes, the difference between the string 100 and 001 is not important. However, the difference between 000 and 110 is. Furthermore, it is easy to notice that the two landscapes (algorithms) differ in their sensitivity to the comparison between 011 and 010.

## 4.2 Analyzing a search algorithm

The performance landscape gives us the difference between the effective distance of any two solutions. This depends on the search operators used by the algorithm. A careful examination of these values can help assess explicitly the effects of the interaction of the search operators.

The coefficients represented by the performance landscape are not precise. They are based on empirical results and therefore they can be noisy. In order to account for the noise in these values, we need to use a clustering algorithm. The clusters give us a robust way to infer the properties of the algorithm. The original values can be used for a finer analysis of the search operators.

In order to demonstrate how this can be done, we used the k-means algorithm to cluster the entries of each of the two performance landscapes described in section 4.1.

Figure 4 gives us the result of the clustering. The tables present the optimal clusters and the value of each entry in the performance landscape. The entries are sorted in ascending order. The curly brackets next to the tables show alternative, sub optimal clusters.

Uniform Crossover			One point Crossover		
Value	Entry		Value	Entry	
0.06	010	100	0.09	011	110
0.21	011	101	0.24	001	100
0.46	001	100	2.7	001	010
0.77	011	110	2.8	101	110
0.82	101	110	2.9	010	100
0.84	001	010	2.95	011	101
5.23	000	100	3.83	000	010
5.49	000	001	6.79	000	100
5.53	000	010	7.46	000	001
10.1	011	100	9.26	011	100
10.5	001	110	9.38	001	110
10.7	010	101	9.63	100	101
12	001	011	9.86	001	101
12.3	100	101	11.8	010	101
12.5	100	110	12	100	110
12.7	010	011	12.8	001	011
12.7	010	110	13.1	000	101
12.7	001	101	17.9	000	011
15	000	110	18.1	000	110
15.6	000	011	19.1	010	011
16.1	000	101	19.4	010	110

Figure 4. The clusters obtained with the k-means algorithm for the performance landscape of GAs using one-point and uniform crossover. The curly brackets represent alternative clusters.

The tables in figure 4 can be used in order to calculate the relative distance of the solutions from one another. Figure 5 plots the relative effective distance of each point from the optimum. For simplicity we do it on a one dimensional graph. Some of the information in the tables is lost. However, in this way we can show the *main* properties in a clear way.

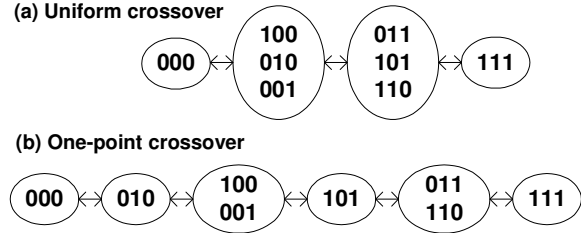


Figure 5. Effective distances for uniform crossover (a) and one point crossover (b).

Figure 5.a gives the results for uniform crossover. It is clear that the effective distance in this scenario is equivalent to the distance defined by the neighborhood structure. Indeed uniform crossover is not biased w.r.t the position of the bits in the string. The distance from the optimum is a good indicator of the probability of producing the optimum by the uniform crossover operator.

A careful examination of the corresponding table in Figure 4 reveals an even more interesting picture. The entries  $c_{001,110}$ ,  $c_{010,101}$ , and  $c_{100,011}$  have lower values than the other entries in the third cluster. This cannot be explained by the neighborhood structure. A possible explanation is the fact that these entries represent a competition (tournament) between complementary (w.r.t the target solution) strings. Since the population is finite and one of the two complementary strings was not chosen to go into the mating pool, the probability that the crossover operator creates the optimum decreases.

Figure 5.b gives the results for one-point crossover. This operator is biased w.r.t bit position. In particular, the strings 010 and 101 cannot create the optimum. This is the reason for the slight difference of the positions for these strings.

The corresponding table in Figure 4 reveals some additional interesting properties in this case as well. Firstly, a smaller effective distance between complementary strings exists here as well. Moreover, notice that the value of  $c_{010,110}$ , for example, is higher than  $c_{000,110}$ . It means that a landscape for which the string 110 wins on 000 but loses to 010 is *more difficult* than the opposite case (110 loses to 000 but wins on 010). This is completely counter-intuitive. However, the string “010” is a possible future competitor of the string “001” (the complementary of “110”). Therefore, when considering more than one generation, this reduces the probability to produce the optimum.

The values of our performance landscape could be further analyzed. However the objective of this paper is to exemplify the way this framework can be used, rather than have a through analysis of any particular algorithm. It is important to emphasize that our technique (i.e. computing the performance landscape and then clustering it) does not assume anything about the algorithm.

## 4.3 Validation of the results

In order to validate the results obtained for the performance landscape we used a simplified version of equation 7

$$c_{i,j} \cong \alpha(p(X_{target} | X_i) - p(X_{target} | X_j)) \quad (8)$$

where we consider only the absolute probability of finding the optimum given a particular string.

In our case, the probability of the algorithm finding the optimum given a particular string can be *approximated* as the probability to

produce the optimum string (111) using the crossover operator. This is true only for one generation. Furthermore, we assume that the probability to select any of the other strings is equal.

$$p(111|x_0) = \sum p_s(x_i)p_c(x_0, x_i) \quad (9)$$

where  $p(111|x_0)$  is the probability to produce the optimal string given that  $x_0$  is chosen in a tournament,  $p_s(x_i)$  is the probability to select  $x_i$  (for the sake of simplicity, we assume that it equals  $\frac{1}{N}$ ) and  $p_c(x_0, x_i)$  is the probability that the crossover operator will produce the string 111.

Table 3 gives us this probability for each point in the search space. In the case of uniform crossover, this probability was calculated for one generation. For one point crossover, it was calculated for two generations. Following equation 8 we can now estimate the importance of each entry in the performance landscape. Entries which belong to the same cluster should have similar importance.

**Table 3. The probabilities of the GA finding the target solution using each of the possible points in the space.**

String	Uniform Crossover	One point crossover	
		First Generation	Second Generation
000	0.015625	0	0.046875
001	0.046875	0.0625	0.09375
010	0.046875	0	0.078125
011	0.109375	0.1875	0.15625
100	0.046875	0.0625	0.09375
101	0.109375	0.125	0.140625
110	0.109375	0.1875	0.15625

This is indeed the case for uniform crossover. The clusters obtained by the k-means algorithm indeed matched the clusters obtained by applying equation 8.

Since this model is very simplified, for the case of the one point crossover the clusters obtained do not show a perfect match. However, the correlation between those values and the values of the entries in the performance landscape (0.95) reveals that most of the clusters can be explained.

In any case, these probabilities account more for general properties rather than for fine ones (see previous section).

#### 4.4 Emergence of neighborhood structure

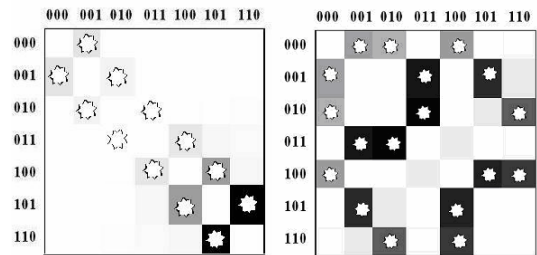
In the previous section we were able to find the properties of a search algorithm using a simple analysis of the performance landscape. Obviously this in turn depends on the neighborhood structure as well. Indeed, the clusters obtained for uniform crossover could give us an idea of the connectivity of the search space (e.g. the distance between 110 and 000 is maximal).

The performance landscape summarizes the contribution of the different components of the algorithm (i.e. operators, neighborhood structure). The objective of this section is to illustrate this, particularly for the neighborhood structure. We

show explicitly that for local search, the results are straight forward. Since in local search some points in the search space cannot be reached from others, certain entries in the performance landscape will be zero (having no relevance to the search).

In order to show this, we found the performance landscapes of a simple local search, with two different neighborhood structures: one based on a one bit flip as a local improvement and one based on a natural representation. We used a simple hill-climber. The performance measure was the number of times the algorithm found the solution in 5000 fitness evaluations. Each time the algorithm reached a local optimum, the algorithm restarted from a different random position. The size of the landscape was 8.

Figure 6 shows how the connectivity of the search space can be predicted by the performance landscape. In order to visualize the landscape structure more easily, we present this time the symmetric part of the performance landscape as well.



**Figure 6. Image representation of two performance landscapes for a hill-climber. The neighborhood structure of the left image was one for natural numbers, that of the right one was hamming neighborhood. The stars illustrate that the neighborhood structure can be reconstructed from the performance landscape**

## 5. Discussion

Developing a theoretical framework for the analysis of a new algorithm is difficult. The constant production of new metaheuristics makes the gap between theory and practice hard to bridge.

In this paper we suggest to use the *performance landscape* as a tool to help the theoretical analysis of new algorithms. It is a *general* tool. It can be applied in the same way to *any* algorithm. It can be used without the need of going through the process of analyzing the specific properties of the algorithm.

The *performance landscape analysis* provides test problems of any degree of difficulty. It gives a framework on which the difficulty of other (real world) problems can be assessed in a robust way. Furthermore, it gives information about the combined effect of the neighborhood structure and the search operators.

Naturally, it is not possible to completely avoid considering the details of the algorithm. In fact, having this knowledge as a starting point makes the analysis much more efficient.

The GA is perhaps one of the most studied metaheuristics. In order to validate our *performance landscape analysis* framework, we gave an example illustrating how it can be applied to the study of GA. We constructed easy and difficult landscapes. Using a static schema analysis we proved that the predicted optimal and worst landscapes coincide with those predicted by traditional GA theory.

The performance landscape gave us information that otherwise could be derived only by applying a complex theory (schema analysis). Furthermore, it gave us this information without having *any* knowledge whatsoever about the algorithm.

It is not possible to compute performance landscapes of big size. Naturally, the larger the landscape is the more likely it is to unfold more of the complexity of the algorithm. The empirical results of this paper were obtained by analyzing a landscape of size 8. Still the results gave many insights about the GA in an automatic way.

We believe that some of our results can be generalized. In particular, the properties of best and worst landscapes of small size could be analyzed and then best and worst landscapes can be constructed for bigger sizes.

Studying the clusters of the performance landscape can produce different approaches for the study of realistic landscapes. Furthermore, the analysis of large landscapes could be performed by focusing only on particular areas on the performance landscape. This can be done by leaving most of the entries constant and varying only the specific areas of interest.

## 6. Conclusions

The study of a new metaheuristic is a long, iterative process. It begins with a basic analysis of the algorithm which gives rise to hypotheses about its properties. This is then compared and contrasted with empirical evidence, which either supports or does not support the hypotheses. If it doesn't, one must start all over again.

Exact properties of the algorithm can be calculated in a probabilistic manner based directly on the search operators. However, this is very complicated, particularly for new and complex metaheuristics.

In this paper we have introduced a powerful tool which can boost the theoretical analysis of new metaheuristics. In particular, it provides:

1. Instances of *true* easy and difficult problems specific to an algorithm.
2. Reference problems that can be used as indicators to the difficulty of other problems.
3. A way to assess the properties of the combined effect of the neighborhood structure and the search operators.

All this can be done in an automatic, simple way - without the need of a lengthy and complicated analysis of the search algorithm. We strongly believe that using this technique as the

first step when analyzing a new algorithm can save valuable research time and effort.

## 7. Acknowledgment

This work was supported by the Anglo-Jewish-Association and the Harold Hyam Wingate Foundation.

## 8. REFERENCES

- [1] C. Blum and A.Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.* 35(3): 268-308, 2003.
- [2] S.Forrest and M.Mitchell. Relative Building Block Fitness and the Building Block Hypothesis. In D. Whitley (ed.) *Foundations of Genetic Algorithms 2*. Morgan Kaufmann, San Mateo, CA, 1992.
- [3] Stefan Droste: Analysis of the (1+1) EA for a Noisy OneMax. *GECCO (1)* 1088-1099, 2004.
- [4] A.H.Right, J.E.Row, J.R.Neil. Analysis of the Simple genetic Algorithm on the Single-peak and Double-peak Landscapes. In *Proceedings of the 2002 Congress on Evolutionary Computation CEC 2002*. Hawaii, USA, pages 214-219, 2002.
- [5] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Morgan Kaufmann, 1989.
- [6] Stefan Droste, Thomas Jansen, Ingo Wegener: On the analysis of the (1+1) evolutionary algorithm. *Theor. Comput. Sci.* 276(1-2): 51-81, 2002.
- [7] Thomas Jansen, Ingo Wegener: On the Choice of the Mutation Probability for the (1+1) EA. *PPSN 2000*: 89-98, 2002.
- [8] C.Höhn and C.R.Reeves (1996) The crossover landscape for the onemax problem. In J.Alander (Ed.) *Proceedings of the 2nd Nordic Workshop on Genetic Algorithms and their Applications*, University of Vaasa Press, Vaasa, Finland, pages 27-43, 1996.
- [9] H. Asoh and H. Muehlenbein. On the mean convergence time of evolutionary algorithms without selection and mutation. In *PPSN III*, volume 866 of *Lecture Notes in Computer Science*, pages 88-97. Springer-Verlag, 1994.
- [10] Vose, M.D. *The Simple Genetic Algorithm*. MIT Press, Cambridge, 1999
- [11] Riccardo Poli, Christopher R. Stephens, Alden H. Wright, and Jonathan E. Rowe, "On the Search Biases of Homologous Crossover in Linear Genetic Programming and Variable-length Genetic Algorithms", *GECCO*, Morgan Kaufmann, 2002.
- [12] C.-Y. Lee, E. K. Antonsson: Variable Length Genomes for Evolutionary Algorithms. *GECCO 2000*: 806, 2000.
- [13] Anabela Simões, Ernesto Costa: Parametric Study To Enhance The Genetic Algorithm's Performance When Using Transformation. *GECCO 2002*: 697, 2002.
- [14] Franz Rothlauf. *Representations for Genetic and Evolutionary Algorithms*. Springer 2002.
- [15] Y.Borenstein and R.Poli. Information landscapes. In *Proceedings of GECCO*, ACM, 2005.
- [16] J.H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to biology, control and artificial intelligence*. MIT Press, 1998.
- [17] Y.Borenstein and R.Poli. Fitness distribution and GA hardness. In *Proceedings of PPSN*, Springer, 2004.