ABSTRACT

Information Storage Capacity of Genetic Algorithm Fitness Maps

George D. Montañez, M.S.

Chairperson: Gregory J. Hamerly, Ph.D.

To accurately measure the amount of information a genetic algorithm can generate, we must first measure the amount of information one can store, using a fitness map. The amount of information generated, minus the storage capacity, gives a tighter estimate on the levels of information generated by genetic algorithms.

To measure the information storage capacity of fitness maps, we use the method suggested by Abu-Mostafa *et al.* (Abu-Mostafa and St Jacques, 1985) for measuring the information storage capacity of general forms of memory. Additionally, we measure the information in reference to the *active information* metric, as developed by Dembski *et al.* (Dembski and Marks, 2009b). Our results show that a number of bits linear in the size of the search space can be stored in a fitness map, but only a logarithmic number of bits can be extracted by a genetic algorithm with stabilizing population and fixed population size.

Information Storage Capacity of Genetic Algorithm Fitness Maps

by

George D. Montañez, B.S.

A Thesis

Approved by the Department of Computer Science

_____

Donald L. Gaitros, Ph.D., Chairperson

Submitted to the Graduate Faculty of
Baylor University in Partial Fulfillment of the
Requirements for the Degree
of
Master of Science

Approved by the Thesis Committee

_____

Gregory J. Hamerly, Ph.D., Chairperson

_____

Eunjee Song, Ph.D.

_____

Robert J. Marks II, Ph.D.

Accepted by the Graduate School
August 2011

_____

J. Larry Lyon, Ph.D., Dean

Page bearing signatures is kept on file in the Graduate School.

Thesis Corrections

George D. Montañez

January 1, 2014

1 Corrections

Lemma 7 contains an error, since the starting inequality does not hold. The terms do not have the same number of terms, and therefore, the rest of the proof does not follow.

Lemma 8 relies on Lemma 7, and thus does not hold either.

These two lemmata are used to prove Theorem 3, and only Theorem 3. Thus, Theorem 3 should not be considered proven. Theorem 3 states:

Theorem 3. *For $b \geq 0$, The number of target sets providing b or more bits of active information in a set S_2 of all possible target sets is less than or equal to $\frac{|S_2|}{2^b}$. The null search probability of finding such a target set in S_2 is, therefore, less than or equal to 2^{-b}.*

A slightly weaker version of Theorem 3 is proven in [1], for $b \geq 2$ and $n \geq 19$, where $n = \log_2 |S_2|$, which is the size of each function in bits. The theorem statement is reproduced here:

Theorem 1. *For $b \geq 2$ and reasonably sized search spaces (of size $n \geq 19$), the number of functions (target sets) providing b or more bits of active information in a finite set S_2 of all possible fixed-length target functions of size n is less than or equal to $\frac{|S_2|}{2^b}$. The probability of finding any such target set in S_2 under uniform random sampling is, therefore, less than or equal to 2^{-b}.*

References

[1] George D Montanez. Bounding the number of favorable functions in stochastic search. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 3019–3026. IEEE, 2013.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This volume would not be possible without the work and encouragement of Robert J. Marks II, Bill Dembski and Winston Ewert. In addition, the input and constructive comments given by Dr. Greg Hamerly were essential to the improvement and completion of this work.

DEDICATION

To

My mother

My wife Luz

My mentor Bob Marks

and Jesus,

My king and redeemer.

Gal. 6:14

CHAPTER ONE

Introduction

How much information can a genetic algorithm generate? Answering this question requires one to first determine how much information a genetic algorithm can store, since examining the output of a device may not reveal how much information the device actually generates. If the device is similar to a memory system or storage drive, the output may consist entirely of information that was previously stored on the device, not generated *de novo*. Genetic algorithms produce measurable results (Floreano and Keller, 2010) and quantifiable amounts of information (Schneider, 2000). However, failure to quantify the information stored in such algorithms prevents one from making strong claims regarding their ability to generate information. Therefore, the focus of this present study is to quantify the amount of information that can be stored in the fitness map of a genetic algorithm.

1.1 Main Results

In this study, we find that a fitness map defined over a set of elements Ω can store a number of bits that is linear in the size of Ω. Furthermore, we find that the number of bits one can extract from the output of a population based method with fixed population size, such as a genetic algorithm, is logarithmic in the size of Ω, for any fitness map resulting in a stabilizing population. This logarithmic information extraction bound is also derived independently using Dembski and Marks' *active information* metric (Dembski and Marks, 2009b).

In addition, we find that the number of fitness maps that allow for maximum information extraction is less than or equal to $\frac{|\Omega_2|}{|\Omega|}$, where $|\Omega_2|$ is number of fitness maps possible given a search space Ω and a finite range of possible fitness values. It is demonstrated that finding a fitness map for a fixed target that produces maximal

active information is at least as difficult as the original search problem, and that the same holds for selecting a target set for a fixed fitness map. In general, it is demonstrated that selecting either a fitness map or a target set that produces b or more bits of active information requires at least b bits of information.

Simulations of genetic algorithm search problems are performed, which empirically confirm many of the theoretical results proven and estimate the distribution of fitness map performance over the space of possible fitness maps. From these, we find that the mean probability of success for a set of uniform-randomly sampled fitness maps is equal to the probability of success of the original search (under uniform random sampling), and that a set of uniform-randomly sampled target sets has roughly the same mean. The similar means of performance suggest that selecting a good fitness map for a target is as difficult as selecting a good target for a fitness map, with both problems being at least as difficult as the original search problem.

Taken together, these results support conservation of information theorems (English, 2004; Wolpert and Macready, 1997; Schaffer, 1994; Dembski and Marks, 2009a; Schumacher et al., 2001) and suggest that success in genetic algorithm search can be made possible by the information storage capacity of genetic algorithms, being exploited through programmer choice in fitness map selection and definition of target sets.

The remainder of this chapter will introduce the relevant concepts of genetic algorithms and fitness maps, and further define the specific problem under consideration.

1.2 Genetic Algorithms

Genetic Algorithms are search and optimization systems inspired by biological evolutionary processes that have the following general structure, following (Bäck and Schwefel, 1993):

(1) *Initialize* population of individuals, $P(0)$, where 0 represents the current time step.

(2) *Evaluate* initial fitnesses of population (according to **fitness function**.)

(3) While the *termination* condition is not met by current population, $P(t)$:

- *Recombine* members of the population (according to **recombination operator**.), forming temporary population, $P'(t)$.
- *Mutate* members of $P'(t)$ (according to **mutation operator**), forming temporary population $P''(t)$.
- *Evaluate* fitness of all members of $P''(t)$ (according to **fitness function**.)
- *Select* members from $P''(t)$ for inclusion in next generation (according to **selection operator** that uses fitness evaluations calculated in previous step.) Set $P(t+1)$ equal to the set of these selected members with possible additional members copied from the previous generation, $P(t)$.

Each of the italicized items represents a point of action where many different methods and parameters can be used. The items in bold represent small sub-methods used by the algorithm for carrying out some of the various action items.

Genetic algorithms can vary in their specific implementations as well as in their usage (Goldberg, 1989; Reeves and Rowe, 2002). Some implementations eschew recombination altogether, while others make it the primary method of producing variation. Since the term "genetic algorithm" covers several distinct variations on the above theme, we limit our study to one specific implementation, which we call a genetic algorithm of type G_a. In general terms, a G_a genetic algorithm follows the outline above, and represents a standard implementation of a genetic algorithm. It is defined as follows:

Definition 1.2.1 (Genetic Algorithm G_a). A *genetic algorithm of type* G_a is any genetic algorithm using a population of k binary strings of length m initially selected, with uniform probability, from a search space Ω of all binary strings of length m, subject

to uniform crossover (Sywerda, 1989), mutation with gene-wise mutation rate μ, and fitness proportional roulette wheel selection (De Jong, 1975) based on fitness values contained in fitness map f. The population size remains fixed at k for the duration of the search.

Chapter 7, Section 7.3, provides detail on the implementation of the various operators within this specific type genetic algorithm. Reeves *et al.* (Reeves and Rowe, 2002) also describes various methods of selection (including roulette wheel selection), and variations on recombination, such as uniform and generalized n-point crossover.

Having determined a specific genetic algorithm model, we now discuss the basic search problem of this study. Let $\Omega = \{I \mid I \in \{0,1\}^m, m \in \mathbb{N}\}$ denote a finite *search space* composed of elements (referred to as individuals) encoded as fixed-length binary strings. From this search space, a subspace T is specified as the *target*, where $T \subseteq \Omega$ and $T \neq \emptyset$. The goal of the search is to produce a population of individuals that include elements in T, starting from a randomly selected initial population. Some genetic algorithm searches are concerned with maximizing the average fitness of a population (Goldberg, 1989, pg. 30), while others are concerned with locating populations that include at least some individuals with desired traits (Floreano and Keller, 2010; Schneider, 2000). Here, we are concerned with a problem of the second kind, namely, selecting some element from Ω belonging to set T.

1.3 Fitness Maps and Fitness Functions

A *fitness function* is any function $\Phi : I \to \mathbb{R}$ that maps an individual, I, of a population to a real number (Bäck and Schwefel, 1993). To simplify matters, we consider here only fitness functions that map individuals to binary numbers, $\Phi : I \to \{0,1\}^l, l \in \mathbb{N}$, and consider only finite domains of individuals.

For our finite domains, a *fitness map* is an exhaustive, uncompressed representation of the mapping between individuals and fitness values generated by a fitness

function. It can be defined as the set of all ordered pairs $(i, \Phi(i))$, where i denotes an individual in our domain and $\Phi(i)$ its fitness value.

1.3.1 Fitness Map Representations

Finite fitness maps can be represented as a set of point-value pairs (which we denote as *list form*) or as a one-dimensional array of values (*array form*), where position in the array corresponds to the index of an individual. If each individual is encoded using $\log_2 |\Omega|$ binary digits, then the list form representation ostensibly requires an additional $|\Omega| \log_2 |\Omega|$ binary digits to encode compared to array form, since an identifier for each individual must be encoded along with its fitness value. Although the array form appears to require fewer binary digits overall, it assumes knowledge of the ordering used to index individuals. Selecting one ordering from the $|\Omega|!$ possible requires $O(|\Omega| \log |\Omega|)$ bits of information,[1] so both representations are actually equivalent in the number of binary digits required to encode a single map.

We therefore assume an array form representation, so that a finite fitness map is represented as a binary string of length $n = l|\Omega|$, where l is the number of binary digits required to encode each fitness value (English, 2000). We assume a lexicographic ordering of elements, thereby storing the additional $|\Omega| \log_2 |\Omega|$ bits of identifier information in the algorithm used to decode the fitness map, and not in the fitness map itself.

1.3.2 Fitness Value Representations

If the distribution of fitness values in a fitness map is known beforehand, a compression scheme such as Huffman encoding can be used to assign short binary strings to frequently occurring fitness values and longer strings to less frequently occurring values. If distribution information is not available, however, we assume

[1] Although a lexicographic ordering of the elements in Ω is usually assumed, it is not the only ordering possible for binary strings. Other orderings, such as Binary Reflective Gray codes (Reeves and Rowe, 2002, pg. 99), produce a different interpretation of the positions in the array. Therefore, we must specify which ordering is being used.

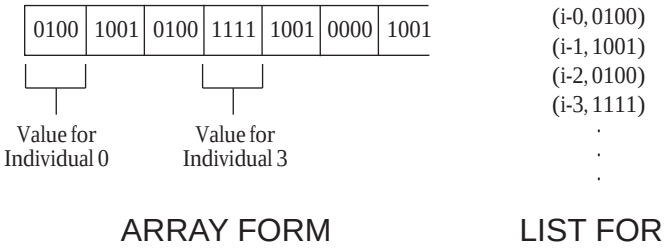

Figure 1.1: Two alternative representations of fitness maps.

an equiprobable distribution of values, so that each value in the set $\mathcal{Y}$ of possible fitness values has a $\frac{1}{|\mathcal{Y}|}$ probability of occurring (Dembski and Marks, 2009a). This assumption results in an encoding length of $l = \log_2 |\mathcal{Y}|$ binary digits per value, following Shannon's entropy formula (Shannon, 1948).

Given an equiprobable occurrence of fitness values, arbitrarily assigning one or more values a shortened binary code does not reduce the number of binary digits required to represent a fitness map, once all information costs are taken into account. This is proven in Section 7.1 (Lemma 1). We therefore assume fixed-length binary fitness value encodings for our fitness maps, allowing for the array form binary string representation discussed in Section 1.3.1.

1.4 Motivation

Having described the model of genetic algorithm and fitness map under consideration, we return to the motivating purpose of this study.

The quantity of information storable in a finite fitness map is bounded above by the number of binary digits required to encode it or, alternatively, the number of bits required uniquely select it from a closed under permutation set of fitness maps, which is an equivalent number of bits. However, the quantity of information that can be extracted from such a map may be less than this maximum, depending on the method of extraction (Ewert et al., 2010). The search process of a genetic algorithm

makes use of a fitness map and extracts information from it to produce a resulting population, which may itself encode information.

To accurately measure the amount of information a genetic algorithm can generate, we must first measure the amount of information a genetic algorithm can store, such as in a fitness map used by the algorithm. Measuring the amount of information generated, minus the storage capacity, allows for a tighter estimate on the levels of information generated by genetic algorithms (Schneider, 2000; Montañez et al., 2010).

1.5 Previous Work

Dembski *et al.* (Dembski and Marks, 2009a,b) have developed information metrics for measuring the information contributions of algorithm selection in general. Research in the area of conservation of information (Wolpert and Macready, 1997; English, 2000, 2004; Schumacher et al., 2001) has examined the gains in information made possible by algorithm bias. In the field of genetic and evolutionary algorithms, most analytical research has focused on applications of schema theory (Holland, 1975; Altenberg, 1995; Radcliffe, 1997), studies of landscape topologies and their effect on genetic algorithm convergence (Jones and Forrest, 1995; Altenberg, 1997; He et al., 2007), and dynamical systems and Markov chain models of genetic algorithms (Vose, 1999; Van Nimwegen et al., 1997). To our best knowledge, no prior work has explicitly and quantitatively measured the information storage capacity of fitness maps, although prior work in the area of conservation of information (English, 2000; Dembski and Marks, 2009a) has made the analysis in this study possible. Research into the ability of genetic algorithms to mine information from fitness oracles (Ewert et al., 2010; Montañez et al., 2010) provided initial motivation for this line of inquiry.

CHAPTER TWO

Information Storage Bounds for a Fitness Map

2.1 Introduction

Let $f : \mathcal{X} \to \mathcal{Y}$ be a fitness map, represented as a binary string, mapping a space of elements[1] $\mathcal{X}$ to a finite set of integer values $\mathcal{Y}$. Following Abu-Mostafa and St. Jacques (Abu-Mostafa and St Jacques, 1985), we define the *information capacity*, C, of a memory to be the logarithm of the number of cases it can distinguish between. To assess the information storage capacity of a fitness map, we consider two situations:

(1) First, when we have access to the fitness map itself and can directly observe the numerical values of f.

(2) Second, when we do not have access to f, but only to a population of k items chosen from $\mathcal{X}$ (with replacement), at time t.

In the second situation, it is assumed that the population becomes stable after some number of steps, so that the frequencies of items in the population stops changing (within a small range of variation, large enough to account for the stochastic effects of a positive mutation rate). If populations are allowed to fluctuate indefinitely, the information channel capacity is infinite, since a simple pattern of fluctuation between two elements can function as simple a binary channel, producing a binary value at each time step, depending on which item is dominant in the population at that time. Therefore, we restrict consideration in the second case to stable populations of fixed size.

[1]In this chapter, the more typical $\mathcal{X}$ is used to represent a space of elements, in order to avoid confusion, since Ω has a specific reserved meaning when discussing asymptotic bounds. The symbol Ω is used in other sections to denote the underlying search space, as a way of preserving continuity with previous work. (Dembski and Marks, 2009a,b)

Our analysis demonstrates that a fitness map can store $O(|\mathcal{X}|)$ bits of information and that, following the conditions outlined, only $O(\log|\mathcal{X}|)$ bits of information can be recovered from a stable population of fixed size.

2.2 First Case: Directly Accessing f

If the numerical values of f are directly accessible, and f is encoded as binary string of length n, a total of 2^n different fitness maps can be distinguished. Using the previously given definition of information capacity, the storage capacity of fitness map f is the expected $\log_2(2^n) = n$ bits.

Given that there are $|\mathcal{Y}|^{|\mathcal{X}|}$ unique fitness maps possible given $\mathcal{X}$ and $\mathcal{Y}$

$$\begin{aligned} C &= \log_2(|\mathcal{Y}|^{|\mathcal{X}|}) \\ &\in O(|\mathcal{X}|\log|\mathcal{Y}|). \end{aligned} \tag{2.1}$$

When $\mathcal{Y}$ is fixed, we have $C \in O(|\mathcal{X}|)$.

2.3 Second Case: Values of f not Directly Observable

In the case where numerical fitness information is not directly available, one must infer fitness information indirectly based on relative reproductive success of items in a population. Information can be encoded within a fitness map f, then extracted by observing the composition of a population at some time t, after the population has stabilized.

In observing stable populations of size k, there are exactly $\binom{|\mathcal{X}|+k-1}{k}$ unique populations possible, following the formula for combinations with repetition. This number of distinguishable cases denotes the number of possible "messages" that can

be sent using this communication channel. Substituting, we get

$$
\begin{aligned}
C &= \log_2 \binom{k + |\mathcal{X}| - 1}{k} \\
&= \log_2 \left(\frac{(k + |\mathcal{X}| - 1)!}{k!(|\mathcal{X}| - 1)!} \right) \\
&= \log_2 \left(\frac{\prod_{i=|\mathcal{X}|}^{k+|\mathcal{X}|-1} i}{k!} \right) \\
&= \log_2 \left(\prod_{i=|\mathcal{X}|}^{k+|\mathcal{X}|-1} i \right) - \log_2(k!) \\
&= \log_2(k + |\mathcal{X}| - 1) + \log_2(k + |\mathcal{X}| - 2) + \ldots + \log_2(|\mathcal{X}|) - \log_2(k!) \\
&\geq (k + |\mathcal{X}| - 1 - |\mathcal{X}|) \log_2(|\mathcal{X}|) - \log_2(k!) \\
&= (k - 1) \log_2(|\mathcal{X}|) - \log_2(k!) \qquad (2.2) \\
&\in \Omega(\log |\mathcal{X}|) \text{ for fixed } k \qquad (2.3)
\end{aligned}
$$

which is an asymptotic lower bound[2] on C. This lower bound corresponds to the case in which we only know that a single element, of all elements in $\mathcal{X}$, has taken over the population and are told which element this is; specifying a single element in $\mathcal{X}$ requires $\log_2(|\mathcal{X}|)$ bits of information.

A second way exists to look at the problem, which provides an upper bound on the information capacity C. We notice that each element of $\mathcal{X}$ takes over some portion of the population in the limit, between 0.0 and 1.0, with all portions summing to unity. If we assume that this percentage can be known to only a finite precision, the problem becomes one of assigning s individual percentage points among the elements of $\mathcal{X}$, with repetition. For example, if each element can have 0 to 100 shares of the population, we must assign the one hundred shares among the $|\mathcal{X}|$ elements of $\mathcal{X}$ in some fashion. In this case, we must count the number of ways there are to distribute 100 items among $|\mathcal{X}|$ buckets (with repetition, taking order into account),

[2]Note, the quantity on line 2.2 is of the order $\log |\mathcal{X}|$, since k is taken as a constant and growth is measured in the size of $\mathcal{X}$. Therefore, the term $\log_2(k!)$ becomes a constant and is ignored asymptotically. Furthermore, $\log_2(k!) \in O(k \log k)$ and $|\mathcal{X}|$ is almost certainly larger than k.

which is $|\mathcal{X}|^{100}$. More generally, there are $|\mathcal{X}|^s$ ways to assign s items among $|\mathcal{X}|$ elements. However, by taking order into account some assignments become repeated in this set, due to the fact that our s items are actually indistinguishable. Therefore, we arrive at an upper bound, since the true number of distinguishable cases (without regard to assignment order) is less than or equal to this number of cases. Taking the logarithm base 2, we find

$$\begin{aligned} C &\leq \log_2(|\mathcal{X}|^s) \\ &= s \log_2(|\mathcal{X}|) \\ &\in O(\log |\mathcal{X}|) \text{ for all fixed } s \end{aligned} \tag{2.4}$$

which is an upper bound for the for the amount of information that can be extracted from a population at time t based on the relative frequencies of elements within the population.

In the case of stochastic genetic algorithms, where population composition is randomized and not deterministically chosen, there will exist some variability in the exact composition of a population at any time step t. However, in this case a sufficient number of runs can be performed using the fitness map f so that the expected relative frequencies of elements in $\mathcal{X}$ can be estimated to a desired precision. Doing so does not change the number of possible distinct populations, therefore not affecting the bounds given.

Since both the lower and upper bounds are of order $\log |\mathcal{X}|$, we have a tight bound of $\Theta(\log |\mathcal{X}|)$ on the amount of information that can be recovered from a genetic algorithm having a stabilizing population, with $O(|\mathcal{X}|)$ being the number of bits that can be stored directly in a fitness map for a fixed precision of $\mathcal{Y}$.

2.4 Discussion

Our analysis demonstrates that while a fitness map allows for up to $O(|\mathcal{X}|)$ bits of storage, only $O(\log |\mathcal{X}|)$ bits can be recovered by a genetic algorithm with a fixed

population size and stabilizing population. Therefore, a great deal of information is lost in using a fitness map as an information storage device within a genetic algorithm. Far less information can be recovered from the population of a genetic algorithm than can be encoded in the fitness map, under the conditions outlined.

It should also be noted that ordinal (ranking) information in isolation, without access to corresponding numerical fitness values, can be demonstrated to function as a simple communication channel (see Appendix A), as is suggested by our analysis of the second case (dealing with unobservable fitness values).

2.4.1 Compression

Although we have considered fitness maps as simple binary strings mapping fitness values to elements in $\mathcal{X}$, this linear representation is rare in genetic algorithms, which compute, rather than store in linear form, fitness information. The reasoning is simple: if you have a search space consisting of binary strings of length 512, there are 2^{512} such strings, roughly 10^{154}; yet there are only an estimated 10^{80} atoms in the observable universe. Therefore, we cannot represent such maps in their entirety within computer memory. Generating fitness information allows us to circumvent this difficulty by not having to store fitness information for strings in our space.

Fitness functions form a compressed representation of a fitness map. Take, for example, a Hamming distance fitness function, which calculates the Hamming distance between two strings of any finite length and can be represented as follows

```
hamming(a, b, length):
    difference = 0
    for i from 0 to length:
        difference += (a[i] != b[i])
    return difference
```

This pseudo-code, while not as compact as possible, is orders of magnitude more compressed than an uncompressed fitness map for all strings of length 512, yet computes the same function. This compression raises a difficulty when attempting to measure the amount of information necessary to represent arbitrary fitness mappings; it would appear some fitness maps are greatly compressible and, therefore, require much less than $|\mathcal{X}| \log_2 |\mathcal{Y}|$ bits of information to encode.

Although true, this compression is not possible in general, since incompressible strings of every length exist.[3] Therefore, to be able to represent an arbitrary fitness map over elements in $\mathcal{X}$ a minimum number of binary digits is required, which is of order $|\mathcal{X}| \log |\mathcal{Y}|$.

[3]Although one could imagine a short program outputting a string representing all binary strings of a given length, one must still split the resulting output into individual strings, which can be done in many possible ways. Selecting one particular split from the set of possibilities, therefore, incurs an additional information cost that must be accounted for. Although an extensive investigation of compression is beyond the scope of this manuscript, the simple point that not all strings are compressible suffices here for our purposes.

CHAPTER THREE

Measuring Information Extraction Through Active Information

3.1 Introduction

Having found bounds for both the information storage capacity of a fitness map and the amount of information that can be extracted from the output of a population based algorithm, we consider a second method of assessing the information extraction capacity, based on *active information*.

Active information (Dembski and Marks, 2009b) is a metric for quantifying improvement in a search over a baseline search method. As originally developed, it uses uniform random sampling with replacement as the baseline search algorithm and considers the performance of a second method, with probability of success q, in comparison to the baseline performance. Let p be the probability of success for a single query taken uniformly at random over search space Ω, where success is defined as selecting an element from Ω belonging to the target set T. The *endogenous information* is defined as $I_\Omega = -\log_2(p)$. This represents the difficulty of the original search problem, in bits. The *active information* I_+ is defined as (Dembski and Marks, 2009b)

$$I_+ = -\log_2\left(\frac{p}{q}\right). \tag{3.1}$$

Active information measures the difference of performance in bits, which correlates to an effective reduction (or increase) in the size of the underlying search space. For example, the search for a single 16 bit binary string over the space of all 16 bit strings has a baseline probability of success $p = \frac{1}{2^{16}}$. Assume an improved search method exists that raises the probability of success to $q = \frac{16}{2^{16}}$. The resulting active information, $I_+ = 4$ bits, represents the reduction in the size of the search problem: the probability of success q is equal to the probability of drawing uniformly

at random a specific 12 bit string from the space of all 12 bit binary strings. The 4 bits of active information have reduced the problem from a 16 bit search problem to an equivalent 12 bit search problem.

We can therefore use active information as an additional method of quantifying the information produced by a genetic algorithm. The active information corresponds to the effective reduction in search problem size, and therefore, to the amount of information produced by the genetic algorithm. Our results in this chapter demonstrate that a maximum of $O(\log |\Omega|)$ bits of active information can be extracted from a fitness map during a single search, which agrees with the information bounds given in Chapter 2. In addition, we find that the number of fitness maps in Ω_2 producing the maximal amount of active information is less than or equal to $\frac{|\Omega_2|}{|\Omega|}$, making the problem of locating such a fitness map at least as hard as the original search in Ω.

3.2 Active Information of Fitness Maps

In Chapter 7, Section 7.3 we demonstrate that genetic algorithms of type G_a are *unbiased search algorithms*, meaning they have performance equivalent to uniform random sampling in the absence of differentiating fitness information. Without fitness information to differentiate one element in Ω from another, a G_a simply performs a random walk through the search space and shows no performance improvement over uniform random sampling.

Because a G_a is an unbiased search algorithm, the algorithm itself, absent of fitness information, does not change the probability of a target being selected during a search. Instead, information within the fitness map alters the probability. By measuring the active information of a G_a search, we are in essence measuring the information provided by the fitness map being used.

Furthermore, by comparing the performance of a G_a using a non-flat fitness map f to the same algorithm using a flat fitness map,[1] we are able to quantify the amount of information provided by fitness map f towards solving the search problem. By bounding the maximal active information we are able to place an upper bound on the amount of information a fitness map can contribute to a search problem.

3.3 Maximum Active Information

We begin by considering a general black-box search algorithm that chooses a probability distribution over the search space Ω and uses it to select elements for evaluation. Using the definition provided in (He et al., 2007), a black-box algorithm is defined as follows:

Definition 1. (Black-Box Algorithm)

(1) Choose some probability distribution π on $\{0,1\}^n$ and produce a random search point $x_1 \in \Omega$ according to π. Compute $f(x_1)$, where f is the fitness evaluation function.

(2) In step t, stop if the considered stopping criterion is fulfilled. Otherwise, depending on $I(t) = (x_1, f(x_1), \ldots, x_{t-1}, f(x_{t-1}))$, choose some probability distribution $\pi_{I(t)}$ on Ω and produce a random search point $x_t \in \Omega$ according to $\pi_{I(t)}$. Compute $f(x_t)$.

Genetic algorithms of type G_a fall under this definition, if we consider each individual of a population a queried point and allow the fitness map to serve as the fitness evaluation function. To begin, however, we simply consider general black-box algorithms and ask what is the optimal search performance for algorithms of this type.

[1] See Definition 7.3.2, page 41.

3.3.1 Optimal Black-Box Search

Let us consider a single search for elements in a target set T within the search space Ω. Assuming that the initial probability distribution π acts as an oracle and provides complete target location information by placing all probability mass on an element within the target, we can locate an element in T in one query. This is the best performance possible, since an algorithm cannot locate the target without sampling at least one point in Ω.

Given a single-query uniform random sampling probability of success equal to p and an assisted probability of success[2] $q = 1$, we find that the maximal amount of active information is

$$\begin{aligned} I_{+_{max}} &= -\log_2\left(\frac{p}{q}\right) \\ &= -\log_2\left(\frac{p}{1}\right) \\ &= I_\Omega. \end{aligned} \tag{3.2}$$

Therefore, the maximum active information produced by a black-box algorithm on a search for elements in Ω is equal to the endogenous information of the original search. Since this is the maximum active information achievable by *any* black-box search algorithm, this quantity places an upper bound on the active information produced by a genetic algorithm. Therefore, the active information provided by a fitness map during a single search is also bounded above by the endogenous information I_Ω.

3.3.2 Active Information Versus Information Extraction Bounds

Comparing the bound in Equation 3.2 to the information extraction bound presented in Chapter 2, we find agreement between the two bounds. Since p is the

[2]Although the probability of success q is dependent on resource constraints, such as how many queries are allowed, it can be modeled as a single query assisted probability of success by either converting k queries into a single query on a k-fold Cartesian product of Ω (Dembski and Marks, 2009b), or by taking the average number of queries into account when estimating q and p.

uniform random sampling probability of success over search space Ω, we have $p = \frac{|T|}{|\Omega|}$. Substituting into Equation 3.2, we find

$$\begin{aligned} I_{+_{max}} &= I_\Omega \\ &= -\log_2(p) \\ &= \log_2\left(\frac{|\Omega|}{|T|}\right) \\ &\in O(\log|\Omega|). \end{aligned} \tag{3.3}$$

This agrees with our tight bound on the quantity of information that can be extracted from a finite fitness map, $\Theta(\log|\mathcal{X}|)$, with $\mathcal{X}$ serving as an alias for Ω in Chapter 2.

3.4 Endogenous Information of the Fitness Map Space Ω_2

Having found that genetic algorithms can produce a maximum of $O(\log|\Omega|)$ bits of active information during a single search, we now consider how many fitness maps achieve this level of performance and the amount of information required to locate such a map in a higher level space of fitness maps.

We let b denote the active information produced during a search, and let Ω_2 denote some finite set of fitness maps. We assume that Ω_2 is nonempty and is not biased in such a way that it contains only fitness maps of high quality, which induce a high probability of success on a search in Ω. Instead, Ω_2 is "fair" in the sense that it does not contain an unusually large number of high performance fitness maps, nor does it contain an unrepresentative number of low quality fitness maps. We formalize this requirement by making Ω_2 an *unbiased fitness map set*. According to Definition 7.4.1, an unbiased set of fitness maps induces an average probability of success equal to the endogenous probability of success p on Ω for any target set T. The requirement that Ω_2 be unbiased simply ensures that all relevant information costs are taken into account. Without the requirement, one could measure the amount

of information required to locate a maximally performing fitness map in biased set Ω_2, but would then have to additionally measure the amount of information required to locate the biased set Ω_2 within an unbiased superset Ω_2'.[3] Therefore, requiring that Ω_2 be unbiased simplifies the calculation of information costs and ensures that no costs are arbitrarily ignored or inflated.

Following Dembski *et al.* (Dembski and Marks, 2009a), if Ω_2 is unbiased, then no more than $\frac{|\Omega_2|}{2^b}$ of the fitness maps in Ω_2 can produce b or more bits of active information. Therefore, selecting a fitness map that produces at least b bits of active information requires at least b bits of information.[4]

Furthermore, the probability of selecting uniformly at random a map that produces at least $\log_2 |\Omega|$ bits of active information from an unbiased set Ω_2 is

$$\begin{aligned} p_{\Omega_2} &\leq \frac{\frac{|\Omega_2|}{2^{\log_2 |\Omega|}}}{|\Omega_2|} \\ &= \frac{1}{|\Omega|} \\ &= p. \end{aligned} \tag{3.4}$$

From this, we see that probability of selecting a fitness map that produces $O(\log |\Omega|)$ bits of active information is less than or equal to p, when selecting uniformly at random from an unbiased set of fitness maps. In other words, the search for a maximally good fitness map is at least as hard as the original search for a target in Ω.

Recall that no more than $\frac{|\Omega_2|}{2^b}$ of the fitness maps in Ω_2 can produce b or more bits of active information, if Ω_2 is unbiased. Assuming Ω_2 is unbiased, the number of fitness maps in Ω_2 producing the maximal $O(\log |\Omega|)$ bits of active information is therefore less than or equal to

$$\frac{|\Omega_2|}{|\Omega|}. \tag{3.5}$$

[3] See Lemma 5, Section 7.5.2.

[4] $-\log_2 \left(\frac{\frac{|\Omega_2|}{2^b}}{|\Omega_2|} \right) = \log_2(2^b) = b.$

3.5 Unbiased Sets and Channel Capacity

The assumption that Ω_2 is unbiased allows us to place a bound on the number of maximal fitness maps in a higher level space of fitness maps. This assumption also allows the information costs of biasing a set of fitness maps towards a particular target to be quantified and taken into account, since we begin with a set of fitness maps that does not favor any particular element in Ω. Without this requirement, one might simply begin with a set of high performing fitness maps and claim that finding a good map requires no information. However, the amount of information required to uniquely identify a particular fitness map in Ω_2 directly affects the amount of information that can be transmitted through use of such a map. The information bounds given in Chapter 2 rely on the assumption that all combinations of k elements are possible and equally probable in a population. As the number of possible population outcomes is constrained or limited, the corresponding information bounds decrease.

To illustrate this, let us take the extreme example of a fitness map space consisting of a single fitness map that produces a predetermined population outcome with probability near 1. Since all searches result in the same final population,[5] the number of distinct "messages" available through the communication channel is reduced to one. A channel restricted to a single message contains no informational entropy and can therefore transmit no information (Shannon, 1948).

In a similar manner, weighting one fitness map in Ω_2 with a higher probability of being selected than others decreases the information storage capacity, since one population outcome (message) becomes more probable than the rest. This lowers the entropy of the communication channel, which decreases the channel capacity accordingly (Shannon, 1948).

[5]It might be objected that since genetic algorithm searches are stochastic, different outcomes are possible for a single fitness map and starting configuration. This criticism is valid, but we are here considering the average frequency of items occurring in the final population over many trials, which reduces stochastic effects and in the limit will produce a single averaged population outcome.

For the derived information storage capacity and active information bounds to hold, we must assume that Ω_2 is unbiased, so that no element in Ω is heavily favored by the set Ω_2, and is large enough to produce all possible population outcomes with roughly equal probability. Assuming the contrary leads to predictable population outcomes and diminishes the information storage capacity.

Although Ω_2 must be sufficiently large and unbiased, increasing its size beyond the minimum number of fitness maps merely increases the information costs for selecting biased (high performance) subsets of fitness maps (see Lemma 3, Section 7.5.2). Increasing the number of fitness maps in Ω_2 can only increase the channel capacity if it produces a greater number of possible population outcomes.

3.6 Summary

In this chapter we quantified the maximum quantity of information that can be extracted from a genetic algorithm using the metric of active information. We found that the active information bound agrees with the information extraction bound presented in Chapter 2, and is bounded above by the endogenous information on the unbiased set of possible fitness maps, Ω_2. Additionally, it was found that the number of fitness maps in Ω_2 producing the maximal amount of active information is less than or equal to $\frac{|\Omega_2|}{|\Omega|}$, and that the degree of freedom one has in selecting a fitness map from the set Ω_2 affects the channel capacity when using that set.

CHAPTER FOUR

Active Information in Target Choice

4.1 Introduction

The analysis presented in Chapters 2 and 3 demonstrates the ability to store information in a genetic algorithm through selection of a fitness map, much like information is stored for later transmission in a flash memory through the selection of bit patterns. The genetic algorithm search process acts as a noisy communication channel, selecting a message (fitness map) at one end that results in another message (final population outcome) appearing on the receiving end with some high probability. Because different fitness maps result in different population outcomes, one can select (or design) a fitness map for a particular desired outcome or target.

In this chapter we wish to determine the effect of holding the fitness map constant and allowing the target set to vary. Doing so allows us to quantify the amount of active information that arises from choice of target set. We find that up to $O(\log |\Omega|)$ bits of active information can be produced through choice of target set alone, given a fixed fitness map, and that the number of target sets producing this maximal active information is less than or equal to $|\Omega|$. More generally, we demonstrate that selecting a target set producing b or more bits of active information requires at least b bits of information, mirroring the results found in Chapter 3. Overall, our results demonstrate that choice of target set can act as an additional source of active information in genetic algorithm search.

4.2 Analysis

4.2.1 Maximum Active Information in Target Choice

Consider a genetic algorithm of type G_a that performs a search on space Ω using a fitness map f. We define the endogenous probability of success for a search as $p = \frac{|T|}{|\Omega|}$, where T is a variable denoting the target set.

From the definition of active information, we see that the maximum active information occurs when p is minimized and q is maximized ($= 1$). The probability p is minimized when $|T| = 1$, so we find

$$\begin{aligned} I_{+_{max}} &= -\log_2\left(\frac{p}{q}\right) \\ &= -\log_2\left(\frac{\frac{1}{|\Omega|}}{1}\right) \\ &= \log_2|\Omega| \end{aligned} \tag{4.1}$$

which agrees with Equation 3.2 of Chapter 3, as well as the boundary case previously identified by Dembski *et al.* (Dembski and Marks, 2009b). Therefore, given a genetic algorithm with a fixed fitness map, up to $\log_2|\Omega|$ bits of active information can be generated during a search on Ω, under the condition that at least one element $\omega \in \Omega$ is, with probability 1, guaranteed to appear in the output population, allowing us to set $T = \{\omega\}$ and fulfill the conditions outlined.

4.2.2 Number of Target Sets Producing Maximum Active Information

Having found the maximum number of bits achievable, we next find how many target sets T, at most, achieve this maximum. We produce two bounds, with the looser bound allowing us to generalize to cases where fewer bits than the maximum active information are produced.

4.2.2.1 *General-form bound.* Theorem 3 states that if S_2 is the set of all possible target sets on Ω, i.e. the power set of Ω with size $|S_2| = 2^{|\Omega|}$, the number of

target sets producing b or more bits of active information is less than or equal to $\frac{|S_2|}{2^b}$. Let τ denote the set consisting of all target sets producing at least b bits of active information. If we set $b = I_{+_{max}} = \log_2 |\Omega|$, we find

$$\begin{aligned} |\tau| &\leq \frac{|S_2|}{2^{\log_2 |\Omega|}} \\ &= \frac{|S_2|}{|\Omega|} \end{aligned} \tag{4.2}$$

which is similar mathematically to the maximum number of fitness maps that provide the maximal $\log_2 |\Omega|$ bits of active information in unbiased set Ω_2 (Equation 3.5). The only difference is that here we select target sets from a higher level space of target sets, whereas in Chapter 3 we selected fitness maps from a higher level space of fitness maps. Keeping this bound in the form $\frac{2^{|\Omega|}}{2^b}$ allows one to easily generalize to cases where less than the maximal active information is produced.

Theorem 3 also states that the probability of selecting uniformly at random a target set providing b or more bits of active information is less than or equal to $\frac{1}{2^b}$. Therefore, the probability of selecting a maximally good target set T from set S_2 uniformly at random is

$$\begin{aligned} p_{S_2} &\leq \frac{\frac{|S_2|}{2^{\log_2 |\Omega|}}}{|S_2|} \\ &= \frac{1}{|\Omega|} \\ &= p \end{aligned} \tag{4.3}$$

Similar to the results of Chapter 3, we find that the search for a maximally good target set is at least as hard as the original search problem in Ω.

4.2.2.2 *Hamerly bound.* A second, asymptotically tighter bound exists for the special case of finding the maximum number of target sets producing the maximum active information. It is based on the reasoning presented in Section 4.2.1, namely, that the active information produced is maximized in the case where the size of the

target set is limited to one element. Since there are $|\Omega|$ such distinct target sets possible, we find an asymptotically tighter bound of

$$|\tau| \leq |\Omega| \tag{4.4}$$

Note that this bound is tighter than the previous upper bound of $\frac{|S_2|}{|\Omega|}$ when $|\Omega| > 3$, since $|S_2| = 2^{|\Omega|}$ and $2^{|\Omega|}$ grows asymptotically faster than $|\Omega|^2$. Setting the inequality

$$\frac{2^{|\Omega|}}{|\Omega|} \geq |\Omega|$$

allows one to easily prove this.

4.3 Discussion

The results in this chapter share similarities with the results of Chapter 3, and reinforce conservation of information theorems in general (Wolpert and Macready, 1997; Schumacher et al., 2001; Schaffer, 1994; Dembski and Marks, 2009b). Even with a fixed fitness map and search algorithm, active information can be generated through the act of defining a target set. The maximum amount of active information gained through this method is equivalent to the maximum amount of active information gained through selection of a fitness map. Furthermore, as was the case in fitness map selection, the search does not gain the active information without cost, but requires at least b bits of information to select a target set producing b or more bits of active information.

These results demonstrate that at least two methods exist for producing active information in a search: defining a fitness map and defining a target set. When one option is not available, the other option may be. Therefore, demonstrating a logarithmic output of information in the size of the search space may not reflect any information generation capability on the part of a genetic algorithm, but may simply demonstrate the extraction of information provided by a programmer through the careful selection of fitness map or target set.

CHAPTER FIVE

Empirical Simulations

5.1 Introduction

Although closed-form results provide the strongest foundation for understanding a system, empirical simulation can serve as a useful guide for discovery and confirmation of analytical results. Here we present results from simulating a genetic algorithm system over a variety of search problems. These results confirm some of the analytical findings from previous chapters, such as showing that G_a performance is equivalent to uniform random sampling in the absence of differentiating fitness information, and that producing b bits of active information by either selecting a fitness map or target set requires at least b bits of information, since uniform-randomly sampling either one provides roughly no improvement of search performance on average. These empirical simulations are therefore helpful in supporting the overall correctness of the theoretical results proven in this study.

5.2 Methods

5.2.1 G_a Code Implementation

We implemented a G_a in the python programming language, with the built-in `random` and `numpy.random` modules being used for pseudo-random number generation. The implementation follows the standard G_a definition, with two slight modifications. First, a resource efficient variant of roulette wheel selection called *stochastic universal selection* (Reeves and Rowe, 2002, pg. 31) is used in lieu of standard roulette wheel selection. This method of selection is similar to roulette wheel selection, but chooses all elements from a population at once, reducing variance in the expected number of offspring per element and speeding up the selection process. Second, a small

constant value of 0.0001 is added to each fitness value, to avoid cases where all fitness values in a population sum to 0, giving undefined results for population relative fitness calculations. Aside from these small modifications, all other aspects of the implementation agree with the G_a definition, namely, a fixed population size of 20 is used during searches, strings are subject to recombination under universal crossover, and elements are mutated with a fixed mutation rate of $\mu = 0.2$.

5.2.2 Search Process

The search process consists of using a G_a to locate a target T in search space Ω, within a given number of queries. The search space Ω is defined as the set of all binary strings of a given length m, with m varying on a per problem basis subject to the constraints $m > 0$ and $m \in \mathbb{N}$. Two different types of target sets are used in individual searches. The *all-ones target* T_{one} consists of one binary string of length m, such that $T_{\text{one}} = \{t \mid t = 111\ldots111, |t| = m, m = \log_2 |\Omega|\}$ and $|T_{\text{one}}| = 1$. This target is used in all searches, except for fixed fitness map searches, where the target set is instead allowed to vary. For those searches, the target set T_{vary} consists of a randomly sized set of strings drawn uniformly at random from Ω.

Each search is run for a limited number of queries, with each element in the population (drawn from Ω) counting as a single query. A search is counted a *success* if a string in the target set is found in the population at any time during the search. We refer to each individual search as a *trial*, since performance is averaged over many trials due to the stochastic nature of genetic algorithms.

5.3 Experiments and Results

To empirically verify the analytical findings of the previous chapters, we performed several experiments using the G_a implementation and search process described in the previous section. We found that in all cases the experimental results agreed closely with the theoretical results, supporting the correctness of the

underlying theory. In this section we describe the experiments performed and their results.

5.3.1 Random Sampling in Ω

As a basis for comparing the performance of a G_a with a flat fitness map to the performance of uniform random sampling with the same number of queries, we used an all-ones target T_{one} of length m, where $m \in \{8, 16, 32\}$, and performed 10,000 trials for each target with a query cutoff of $Q = 1000$. The probability of selecting a single target one or more times from Ω under uniform random sampling in Q queries or less can be calculated analytically, by

$$(1 - (1 - p)^Q)$$

where p is the probability of success for a single query and Q is the maximum number of queries. This translates into the following probabilities for binary strings of length 8, 16 and 32 being selected uniformly at random from the space of all 8, 16 and 32 bit strings, respectively, given $Q = 1000$ queries:

$$\begin{aligned} p_8 &= \left(1 - \left(1 - \frac{1}{2^8}\right)^{1000}\right) && = 0.98 \\ p_{16} &= \left(1 - \left(1 - \frac{1}{2^{16}}\right)^{1000}\right) && = 0.015 \\ p_{32} &= \left(1 - \left(1 - \frac{1}{2^{32}}\right)^{1000}\right) && = 2.33 \times 10^{-07}. \end{aligned}$$

Performing empirical random sampling on Ω using these parameters gave the probabilities of success listed in Table 5.1, averaged over 10,000 independent trials consisting of 1,000 queries each.

5.3.2 G_a Performance Using Flat Fitness Maps

Next, we tested the performance of a G_a using a flat fitness map. We used three different flat fitness maps (f_0, f_1, f_2), each averaged over 1,000 independent trials,

Table 5.1: Results from Uniform Random Sampling

Length (m)	Trials	Queries (Q)	Avg. Probability of Success
8	10,000	1,000	0.980
16	10,000	1,000	0.016
32	10,000	1,000	0.000

with each trial consisting of a maximum $Q = 1000$ queries. For each trial, a fixed all-ones target T_{one} was used. Trials were run for binary strings of length 8, 16 and 32. Table 5.2 shows the results of the experiments. The empirically determined average probabilities of success for the flat fitness maps agree with the expected analytical probabilities of success under uniform random sampling given the same number of queries and the same size of search space. For example, the estimated mean based on all trials for $m = 16$ is 0.0157 ± 0.002, with 95% confidence. This provides evidence that a G_a using a flat fitness map is an unbiased algorithm, as Theorem 2 states, and demonstrates that replication, mutation and selection by themselves provide no benefit to a search absent of differentiating fitness information. However, as we will see in Section 5.3.3, the fitness information must also be accurate in addition to merely differentiating among elements in a search space in order to provide improved search performance.

Table 5.2: G_a Search Performance Using a Flat Fitness Map

Length (m)	Fitness Map	Trials	Queries (Q)	Avg. Probability of Success
8	f_0	1,000	1,000	0.969
8	f_1	1,000	1,000	0.978
8	f_2	1,000	1,000	0.954
16	f_0	1,000	1,000	0.018
16	f_1	1,000	1,000	0.018
16	f_2	1,000	1,000	0.011
32	f_0	1,000	1,000	0.000
32	f_1	1,000	1,000	0.000
32	f_2	1,000	1,000	0.000

5.3.3 G_a Performance Using Randomly Selected Fitness Maps

Our next set of experiments sought to measure the performance of a G_a over randomly selected fitness maps. According to the results in Chapter 3, selecting a fitness map at random and using it to conduct a search should result in performance equal to or worse than uniform random sampling of the original search space, on average. To test this, we used a fixed all-ones target of length $m = 16$ and selected uniformly at random 1,000 fitness maps from the set of all fitness maps over Ω, having integer fitness values in the range [0, 32]. For each fitness map selected, an average was taken over 1,000 independent trials, with each trial having a maximum of $Q = 1000$ queries.

Figure 5.1 shows the performance of all 1,000 randomly selected fitness maps. The distribution is approximately normal, with an estimated mean of 0.0157 ± 0.0002 (95% confidence) and sample standard deviation of 0.004. The estimated mean probability of success is between the analytical and empirical probabilities of success for uniform random sampling using the same number of queries, when $m = 16$. These results, therefore, confirm empirically what was demonstrated in Chapter 3: selecting a fitness map at random and using it to perform a search will result, on average, in performance equivalent to (or worse than) uniform random sampling. Recall that selecting a fitness map that produces b or more bits of positive active information requires at least b bits of information; since we selected a set of fitness maps uniformly at random, instead of using programmer input to bias that set towards our search, we should expect little or no search improvement on average over that set. This is exactly what is found.

Furthermore, given a baseline probability of success $p_{16} = 0.015$ for our search, the maximum active information achievable is roughly 6 bits; yet a fitness map over a search space Ω of 16 bit strings requires at least 65,536 binary digits (uncompressed) to define, since at minimum we must assign a one bit fitness value to each element in

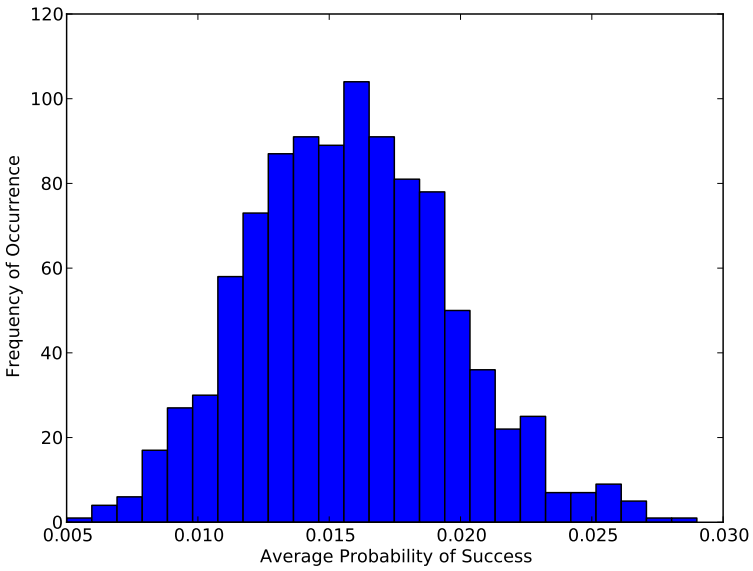

Figure 5.1: Empirical distribution of success probability for randomly selected fitness maps, $m = 16$.

Ω. Therefore, the G_a requires a stored 65,536 bits of information in order to produce a mere 6 bits of active information. Furthermore, these 65,536 bits must be specified, rather than randomly generated, since not all fitness maps result in positive active information. These results are all in agreement with our analytical results.

5.3.4 G_a Performance Using Randomly Selected Target Sets

In order to assess the performance of randomly selected target sets for a fixed fitness map, we conducted two experiments, both on a search space Ω of $m = 16$ bit strings using a Hamming fitness map defined in reference to the all-ones string.[1] This fitness map was used because it has a unimodal landscape topology amenable to genetic algorithm search. In the first experiment, we selected uniformly at random 1,000 subsets of strings from Ω to serve as target sets, and ran 1,000 independent trials with each to measure the average active information produced. In the second experiment, we limited the size of the target set to one and selected uniformly at

[1]Technically, the fitness map was generated by counting the number of ones in a string, which is the fitness map produced by treating Hamming distance from the all-ones string as error, and rewarding minimum error with higher fitness.

random 1,000 strings in Ω to serve as target sets, then conducted 1,000 independent trials with each string to measure the average search performance for a randomly selected target of size 1.

5.3.4.1 *Active information of randomly-sized target sets, T_{vary}.* For the first experiment, the target set T_{vary} was chosen uniformly at random from Ω, such that $0 \leq |T_{\text{vary}}| \leq |\Omega|$. Each set was ran for 1,000 independent trials on the Hamming fitness map, each trial having a maximum query count of $Q = 1000$, and search performance was averaged over all trials. Figure 5.2 shows the distribution of active information for all randomly selected target sets. The maximum active information produced by any sampled target set was 0.002 bits, with a minimum of -0.08 bits and an estimated mean of $-7.23 \times 10^{-5} \pm 0.0002$ bits with 95% confidence. Since the average size of target set was 32,577 (roughly half the elements in Ω), searches often completed in a single generation and the average probability of search success was near 1. However, because the target sets were so large, the average active information produced over all target sets was near zero. Large target sets ensure (with probability near 1) that both uniform random sampling and G_a search find the target given 1,000 queries, thus explaining the lack of active information. The vast majority of target sets sampled (96.9%) produced no active information, with most of the remaining target sets producing near zero bits. These results agree with the theoretical results of Chapter 4, where it was demonstrated that selecting a target set that produces positive active information requires an input of information, leading one to believe that uniform random sampling of target sets should produce 0 or fewer bits of active information, on average. Therefore, the average lack of positive active information serves as confirmation of the results presented in that chapter.

5.3.4.2 *Search performance of fixed-size, randomly selected target sets.* Since large, randomly selected target sets were found to produce no active information in general, we limited the target set to size 1, and selected uniformly at random 1,000

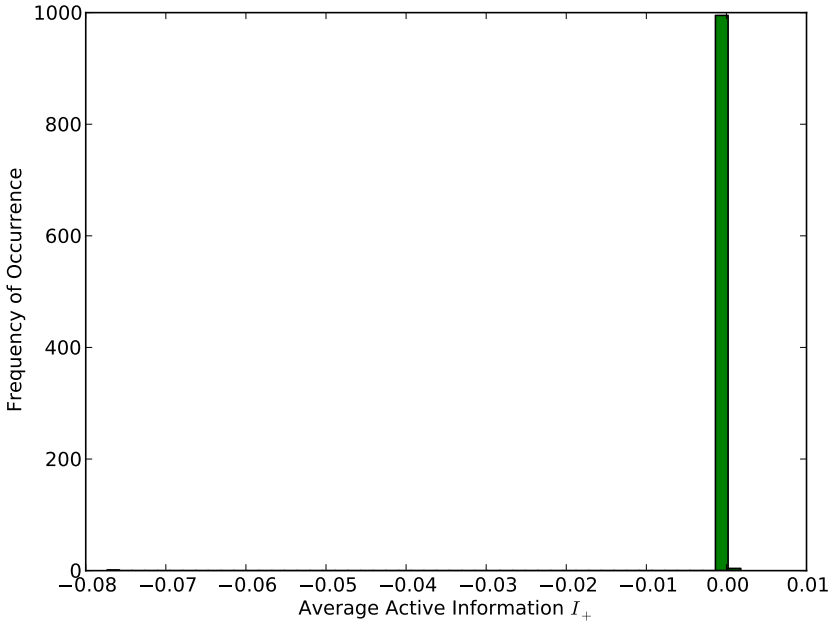

Figure 5.2: Empirical distribution of active information I_+ produced by randomly selected target sets.

target sets of this size. For each target string, 1,000 independent G_a searches were conducted using a query cutoff of $Q = 1000$ queries, and search performance was averaged over all trials. We found that the estimated mean probability of success for all randomly selected target strings was 0.0152 ± 0.0004 (95% confidence), which is near the analytical probability of success for uniform random sampling given the same number of queries, namely $p_{16} = 0.015$, and that the estimated mean active information produced was -0.11 ± 0.038 bits, with 95% confidence. Figure 5.3 shows the distribution of averaged search performance for the uniform randomly selected target sets of size 1, given a fixed Hamming fitness map.

5.4 Conclusion

The simulation results presented here agree with the theoretical results of previous chapters. Taking sample variance into account, we confirmed empirically that a G_a using a flat landscape has search performance equivalent to uniform random sampling, and similarly, that selecting either fitness maps or target sets uniformly at

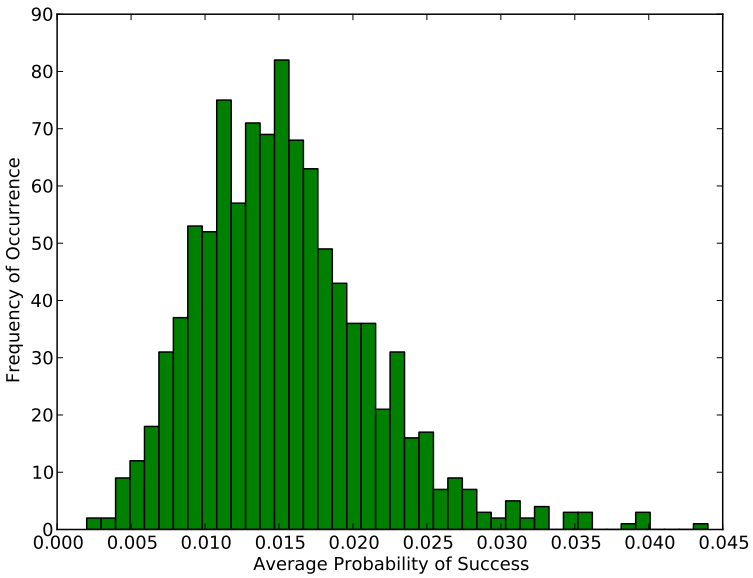

Figure 5.3: Empirical distribution of success probability for randomly selected target sets of size 1.

random results in mean performance equivalent to uniform random sampling. The similar means for fitness map and target set performance suggest that matching a target and fitness map is difficult regardless of which component is held fixed and which is allowed to vary. Furthermore, this agrees with our theoretical results, since producing b or more bits of active information from a G_a search requires at least b bits of information, whether that information is used to select a fitness map or define a target set.

CHAPTER SIX

Conclusion

Determining the amount of information generated by a genetic algorithm requires the prior disclosure and quantification of information sources available to the algorithm. It may be the case that genetic algorithms simply extract and transmit information rather than generate it, much as a telegraph or MP3 player transmits and transforms, but does not generate, information. If, however, the amount of information produced by a genetic algorithm can be shown to exceed the amount of information storable in and available to the algorithm, then a strong case could be made for the ability of genetic algorithms to generate novel information. Therefore, we have sought to quantify the amount of information a genetic algorithm can store in a fitness map, since fitness maps (and functions) are important components of genetic algorithms that greatly affect search performance. As a result, we found that on the order of $O(|\Omega|)$ bits can be stored in a finite fitness map, for a fixed range of fitness values, where $|\Omega|$ is the size of the underlying search space. Furthermore, while a number of bits linear in the size of the search space can be stored in a genetic algorithm's fitness map, only $O(\log|\Omega|)$ bits of information can be extracted from the output of a genetic algorithm with a fixed population size and stabilizing population. The same bound was derived here independently using the active information measure devised by Dembski and Marks (Dembski and Marks, 2009b), further confirming the result.

A number of secondary questions were also explored, such as determining the maximum number of fitness maps in an unbiased set that allow for maximal extraction of information, and quantifying the minimum number of bits required to specify such a map. In answer to the first question, we found that no more than

$\frac{|\Omega_2|}{|\Omega|}$ fitness maps allow for maximal information extraction, where Ω_2 is an unbiased set of fitness maps defined over Ω. In answer to the second, we demonstrated that specifying a fitness map that produces $O(\log|\Omega|)$ bits of active information requires at least $O(\log|\Omega|)$ bits of information, when selecting from an unbiased set of fitness maps.

In addition to this, we found that selecting a target set given a single fitness map could produce as much active information as selecting a fitness map given a single target set. Producing b bits of active information by either method requires b bits of information, such that active information produced as output must first be stored as input, either by selecting a fitness map or selecting a target set. Although the results were derived independently, they agree mathematically and are similar in mathematical form.

Simulations were performed in order to confirm empirically the theoretical results presented in this study. It was found that selecting either a fitness map at random or a target set at random resulted in no search improvement on average, and that a G_a using a flat fitness map also had search performance equal to uniform random sampling. Furthermore, the mean search performance for randomly sampled fitness maps closely matched the mean search performance for randomly sampled target sets, indicating that finding a good target set for a fitness map is just as difficult as finding a good fitness map for a target set.

In conclusion, the information bounds presented here provide a preliminary threshold for demonstrating the information generation capacity of genetic algorithms, at least of the specific type considered in this study. If $O(n)$ represents the maximum number of bits storable in the fitness map of a genetic algorithm and it is demonstrated that the same algorithm produces a super linear number of functional bits on output, then one can quantify a positive level of information production for that algorithm. Failure to specify the threshold prevents any objective claim from being made

regarding the actual information generation capabilities of a genetic algorithm. Therefore, this study represents a step forward in the process of understanding the capabilities and strengths of genetic algorithms.

CHAPTER SEVEN

Theorems

7.1 Compression of Fitness Value Encodings

Lemma 1. *Encoding a set of 2^N distinct strings of length N requires each code to contain at least N binary digits. Using shorter codes for some strings does not reduce the total number of binary digits required to represent the set.*

Proof. We begin with a set α of 2^N distinct binary strings, each of fixed length $N(\geq 1)$, bijectively mapping to another set β of 2^N binary strings, such that the combined length of all elements in α is greater than the combined length of all elements in β. We denote elements of β as 'short codes' or simply 'codes.'

Therefore

$$\begin{aligned} \text{TOTAL}_\alpha &= N2^N \\ \text{TOTAL}_\beta &= \sum_{c\in\beta} len(c) \end{aligned} \tag{7.1}$$

where $len(c)$ is number of binary digits in element c, and

$$\begin{aligned} \text{TOTAL}_\beta &< \text{TOTAL}_\alpha \\ \text{TOTAL}_\beta &= \text{TOTAL}_\alpha - m \\ &= N2^N - m \end{aligned} \tag{7.2}$$

for some $m > 0$.

Since there are 2^N elements in β, at least one code in β must have a length of N or more, as there are only $2^N - 1$ binary strings of length less than N. Therefore,

$$\text{TOTAL}_\beta \geq \sum_{i=0}^{N-1} i2^i + N \tag{7.3}$$

with equality holding for Equation (7.3) when β consists of the shortest codes possible.

Given a measure of the number of binary digits necessary to represent set β (namely TOTAL_β), we must add to this the number of bits necessary to recover our original set α from β, assuming that this can be done using some mapping from β to α. There exists $2^N!$ unique bijective mappings possible between β and α, requiring $\log_2(2^N!)$ bits of information to uniquely identify a particular one.

The total number of binary digits necessary to represent set α using compressed codes β becomes

$$\text{TOTAL}_{\beta+mapping} = \text{TOTAL}_\beta + \log_2(2^N!) \tag{7.4}$$

Assume that $\text{TOTAL}_{\beta+mapping} < \text{TOTAL}_\alpha$ and, therefore, $m > \log_2(2^N!)$. Given this assumption, we have

$$\text{TOTAL}_\beta < N2^N - \log_2(2^N!) \tag{7.5}$$

From Equation (7.3), we find

$$\begin{aligned}
\sum_{i=0}^{N-1} i2^i + N &< N2^N - \log_2(2^N!) \\
((N-2)2^N + 2) + N &< N2^N - \log_2(2^N!) \\
\log_2(2^N!) + 2 + N &< N2^N - (N-2)2^N \\
\log_2(2^N!) + 2 + N &< 2^N(N - (N-2)) \\
\log_2(2^N!) + 2 + N &< 2^N(2) \\
\log_2(2^N!) + 2 + N &\not< 2^{N+1}
\end{aligned} \tag{7.6}$$

which can be easily verified for $N = 2$. Having reached a contradiction, we see that our assumption cannot hold, and therefore $\text{TOTAL}_{\beta+mapping} \not< \text{TOTAL}_\alpha$, completing our proof. □

7.2 Probability of Selecting a Good Fitness Map

Theorem 1. *The number of fitness maps providing b or more bits of active information in a set Ω_2 of possible fitness maps is less than or equal to $\frac{|\Omega_2|}{2^b}$, where Ω_2 is any*

unbiased set of fitness maps that has an average probability of success equal to p, the uniform random sampling probability of success for the original search. The null search probability of finding such a landscape in Ω_2 is, therefore, less than or equal to 2^{-b}.

Proof. In previous work (Dembski and Marks, 2009a), it has been shown that the endogenous information of a search-for-a-search (i.e. a search in Ω_2) is bound below by the active information on the lower level search (a search in lower level search space Ω)

$$-\log_2\left(\frac{|T_2|}{|\Omega_2|}\right) \geq I_+ \tag{7.7}$$

where Ω_2 is an unbiased higher-level space with elements inducing an average probability of success on the lower level search equal to p, $|T_2|$ is the number of elements in Ω_2 that induce a lower level probability of success greater than or equal to q, and I_+ is the active information (Dembski and Marks, 2009b), defined as $\log_2(\frac{q}{p})$.

Let b equal the active information. Then

$$\begin{aligned} -\log_2\left(\frac{|T_2|}{|\Omega_2|}\right) &\geq b \\ \log_2(|T_2|) - \log_2(|\Omega_2|) &\leq -b \\ |T_2| &\leq 2^{\log_2(|\Omega_2|)-b} \\ |T_2| &\leq \frac{|\Omega_2|}{2^b} \end{aligned} \tag{7.8}$$

completing our proof. Rearranging terms, we find

$$\frac{|T_2|}{|\Omega_2|} \leq \frac{1}{2^b} \tag{7.9}$$

□

7.3 Genetic Search as an Unbiased Algorithm

7.3.1 Definitions

Definition 7.3.1 (Unbiased Search Algorithm). An *unbiased search algorithm* is any algorithm that searches a space Ω for target T with probability of success equal to p, the probability for a uniform random sampling search on Ω.

Definition 7.3.2 (Flat Fitness Map). A *flat fitness map* is any fitness map f_c such that $f_c(i) = c$ for all i in Ω, where c is a fixed constant greater than zero. In other words, map f_c assigns the same fitness value, c, to every element in Ω.

Theorem 2. *Any G_a utilizing a flat fitness map is an unbiased search algorithm.*

Our proof will demonstrate that the algorithm begins with a probability of success equal to p, and each step of the algorithm leaves this probability of success unaltered. Therefore, combining the steps in serial order does not alter the probability of success for the genetic algorithm.

7.3.2 Proof

We begin with a genetic algorithm of type G_a (See Definition 1.2.1, pg. 3). Prior to initialization of the search, the probability of success is $p = \frac{|T|}{|\Omega|}$, as this is the baseline probability of the search problem. Without loss of generality, we will define T as

$$T = \{t \in \Omega \mid |t| = n, t_i = 1 \text{ for } 0 \leq i < n\}$$

so that T consists of a single element t in Ω, namely the all ones string of length n. Therefore

$$p = \frac{1}{2^n}$$

According to Definition 1.2.1, the algorithm begins by drawing k elements uniformly at random from Ω. We refer to this as the initialization step, and we will demonstrate that if the probability of uniformly selecting t from Ω is p, then the

probability of selecting t from the population of k elements chosen randomly with uniform probability from Ω is also p.

7.3.2.1 *Initialization step preserves probability.* We will place no constraint on whether or not algorithm G_a selects elements from Ω with replacement or without, and will analyze both cases.

Consider the case where we select (randomly, with uniform probability) k elements of Ω without replacement to form a population *pop*. The probability that t is selected using population *pop* is

$$\begin{aligned}
\Pr[\text{selecting } t \text{ from } pop \text{ and } t \in pop] &= \Pr[\text{selecting } t \text{ from } pop \mid t \in pop] \times \Pr[t \in pop] \\
&= \frac{1}{k}\frac{k}{2^n} \qquad \text{(Hypergeometric trials)} \\
&= \frac{1}{2^n} \\
&= p \qquad (7.10)
\end{aligned}$$

Therefore, the probability of selecting target t from a population of k elements sampled uniformly without replacement from Ω is equal to p.

For the case where replacement is allowed, we have k Bernoulli trials, each with a probability of success equal to $\frac{1}{2^n}$. Let X be a random variable denoting the number of times t is selected from Ω in all k trials.

Therefore

$$\begin{aligned}
\Pr[\text{selecting } t \text{ from } pop \text{ and } t \in pop] &= \frac{\mathbb{E}[X]}{k} \\
&= \frac{k\left(\frac{1}{2^n}\right)}{k} \\
&= \frac{1}{2^n} \\
&= p \qquad (7.11)
\end{aligned}$$

From this, we see that in both cases the probability of success for algorithm G_a is unaffected by the initialization step. Therefore, the probability of success remains p after initialization.

7.3.2.2 *Selection step preserves probability.* Having a population of k elements created by the initialization step and a probability of success still equal to p, we now consider the effect of fitness proportional roulette wheel selection used by G_a when utilizing a flat fitness map. In general, with a non-flat fitness map, selection assists the probability of success by rewarding elements of high fitness with an increased presence in the population, and reducing the number of elements with lower fitness. If some strings are more easily transformed into the target via mutation and recombination, and such strings are assigned a fitness proportional to how easily they can be transformed (as measured by their transformation or edit distance), then over time the probability of success will be increased through the selection process.

Consider, however, the case where all elements in Ω are assigned the same fitness value, as is the case when using flat fitness maps. Under this assumption, all strings have an equal probability of being selected, regardless of their distance from the target. Therefore, fitness proportional selection using the roulette wheel method (De Jong, 1975) becomes equivalent to uniform random sampling of strings in the current population.

To confirm this quantitatively by assessing the effect of fitness proportional roulette wheel selection when using a flat fitness map, we assume that some fraction of the population, equal to $\frac{1}{x}$, consists of low edit distance strings that are likely to be transformed into the target t through later mutation and crossover steps. We define this formally as the set of strings L such that $L = \{s \in pop \mid \Pr[m_{\Theta_m}(r_{uni}(s)) = t] > p\}, |L| = \frac{1}{x}(k)$, where k is the population size, m_{Θ_m} denotes the mutation operator with parameters Θ_m, and r_{uni} denotes the recombination (uniform crossover) operator. Let L_{before} denote the set L prior to the selection step and L_{after} denote the set after application of selection.

Our goal is to assess whether the selection step increases, decreases, or maintains the size of L, or mathematically, which of the following holds: $|L_{\text{before}}| < |L_{\text{after}}|, |L_{\text{before}}| > |L_{\text{after}}|$, or $|L_{\text{before}}| = |L_{\text{after}}|$.

Since fitness proportional selection under a flat fitness map allows each element in the population to be selected with equal probability, the selection step becomes a set of k Bernoulli trials, where each trial is an attempt to select an element in L with probability of success equal to $\frac{1}{x}$. Let X be a random variable denoting the number of elements in the population in L after the selection step is applied. Therefore

$$\begin{aligned} |L_{\text{after}}| &= \mathbb{E}[X] \\ &= k\left(\frac{1}{x}\right) \\ &= |L_{\text{before}}| \end{aligned} \tag{7.12}$$

Therefore, the selection step using a flat fitness map does not change the number of low edit distance strings in the population, having no effect on the expected size of L. Since the number of low edit distance strings in the population is unaltered by selection under these constraints, the probability of success for the G_a algorithm is unaffected by the selection step; if it were p before selection, it remains p after selection is applied.

7.3.2.3 *Uniform crossover step preserves probability.* We now consider the effect of uniform crossover (Sywerda, 1989) on the probability of success. We implement uniform crossover by randomly generating a mask, m, of length n (equal in size to the elements of Ω), where each position in the mask is chosen from the set $\{0, 1\}$ with uniform probability. We then select two strings, labeled p_1 and p_2, uniformly at random from our population of k elements, and create a new string s such that

$$s = m \otimes p_1 \oplus \overline{m} \otimes p_2 \tag{7.13}$$

where $\otimes$ is bitwise multiplication, $\oplus$ is bitwise addition and $\overline{m}$ is the bitwise complement of mask m.

Since none of the previous steps have biased the composition of the population to favor the target or disfavor the target, and since the target t consists of all ones, we can conclude that the ith binary digit of any string in the population is as likely to be zero as it is one. Assume this were not the case, and that the probability of the randomly selected binary digit being one was greater than $\frac{1}{2}$. A bias would then exist towards finding the target t, since the binary digits of strings in our population would be more likely to match the target than not. In this case, the previous steps of the algorithm would have produced a population of strings biased, however slightly, towards being similar to the target. The converse would be true if the probability of finding a one at a randomly selected binary digit were less than $\frac{1}{2}$, in which case there would exist bias against finding the target t. In either case, the probability of success would no longer be p. Therefore, the probability that a randomly selected binary digit uniformly selected from the population equals one must be $\frac{1}{2}$.

Given this fact, we can calculate the probability of the uniform crossover operation producing a string matching the target t. We assume two parents are selected uniformly from the population and we randomly generate a mask m as described above. A new string, s, is then created by uniform crossover. For each position i in string s, the probability that at least one of the parents have a binary digit that matches the target at position i is equal to $1 - (\frac{1}{4})$. There are four possible outcomes, if we consider the two binary digits (one from each parent) at position i: 00,01,10,11. Since the parents are chosen independently and uniformly, and since the probability that a random binary digit chosen uniformly from the population will be one is 0.5, each of these four possibilities is equally probable.

Therefore, for a single position i in s, the probability of having a matching binary digit in a parent and selecting it with mask m is

$$\begin{aligned}\Pr[\text{Matching Digit in Parent(s) and is Selected}] &= \frac{1}{4}(0) + \frac{1}{4}(0.5) + \frac{1}{4}(0.5) + \frac{1}{4}(1) \\ &= \frac{1}{2}\end{aligned}$$

Then the probability that all positions will have a match and be selected correctly by the mask is

$$\begin{aligned}\Pr[\text{All Positions Have Match and are Selected}] &= \frac{1}{2^n} \\ &= p \qquad (7.14)\end{aligned}$$

Therefore, using uniform crossover to sample a new point in Ω by creating a new string from parents chosen at random from the current population, under our constraints, is equivalent to sampling a new string uniformly at random from Ω. If the probability of success before the crossover step is p, then using crossover to select the next point in Ω has a probability equal to p of succeeding. Furthermore, just as independent uniform trials of a search with replacement have the same probability of success for each query, such is the case with uniform crossover. After k queries using crossover, the probability of success remains p if the target has not been found. Therefore, the uniform crossover step does not alter the probability of success for the search.

7.3.2.4 *Mutation step preserves probability.* As none of the previous steps of the algorithm have changed the probability of success for the search, we will now determine if the mutation operator has any effect on the probability of success.

We begin with an arbitrary string selected uniformly at random from our population, which we label s. The probability that a given position i already matches the target is $\frac{1}{2}$, as was demonstrated in the previous step. We consider a mutation operator operating with a per gene mutation rate of μ, meaning that each binary

digit position in s has a μ probability of being mutated (from a 1 to a 0, or a 0 to a 1).

For a single position i in s, the probability that it will match the target *after* application of the mutation operator is

$$\begin{aligned}
\Pr[s_i = t_i \text{ after mutation}] &= \Pr[s_i = t_i \text{ before mutation and no mutation is applied}] \\
&\quad + \Pr[s_i \neq t_i \text{ before mutation and mutation is applied}] \\
&= \Pr[s_i = t_i \text{ before mutation}](1-\mu) \\
&\quad + \Pr[s_i \neq t_i \text{ before mutation}](\mu) \\
&= \frac{1}{2}(1-\mu) + \frac{1}{2}(\mu) \\
&= \frac{1}{2}((1-\mu)+\mu) \\
&= \frac{1}{2} \qquad (7.15)
\end{aligned}$$

Since all positions in s are independent, the probability that all positions will match after mutation is

$$\begin{aligned}
&= \left(\frac{1}{2}\right)^n \\
&= \frac{1}{2^n} \\
&= p \qquad (7.16)
\end{aligned}$$

Therefore, using the mutation operator to sample new points in Ω by selecting uniformly at random a string in the population and mutating it according to the per gene mutation probability is equivalent to selecting a string uniformly at random from Ω. Thus, using the mutation operator to select points does not bias the search toward finding the target or hinder it from finding the target. The probability of success after applications of the mutation operator remains p.

7.3.2.5 *All steps preserve probability of success.* Since we have demonstrated that the algorithm begins with a probability of success equal to p and no step in

the algorithm changes the probability of success, the probability of success after application of all steps remains p. In a genetic algorithm of type G_a, each step feeds into the next step, and the steps are repeated for many cycles (or alternatively, many generations). At any step, the probability of success is p. Therefore, the probability of success after any number of steps is p, completing our proof. $\square$

7.3.3 Discussion

We have demonstrated that none of the steps (initialization, selection with flat fitness map, uniform crossover or mutation) alter the probability of success for the original search. Each step transforms the population in some way; however, none of the transformations are biased towards assisting the algorithm in finding the target. At each step, the probability of success for the search remains unchanged after the step is applied.

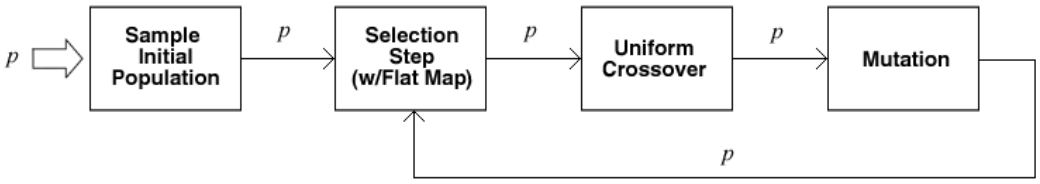

Figure 7.1: G_a operation schematic. Probability of Success Preserved at Each Step. The initial probability of success, p, is transferred at each step to the next operation, so that the operation begins with a probability of success equal to p. At no step is this probability altered.

The above theorem has been proven only for genetic algorithms employing the crossover and mutation operators as defined above. Although the proof has assumed specific use of these two operators, it is highly likely that other operators would produce a similar result, as long as the operators themselves contain no inherent bias towards producing a specific target. Future work may prove this result for a wider range of genetic algorithms.

7.4 Number of Unbiased Sets of Fitness Maps

7.4.1 Definitions

Definition 7.4.1 (Unbiased Fitness Map Set). Let p be the probability of success for a uniform random sampling search on Ω. An *unbiased fitness map set* is any set of fitness maps with an average probability of success equal to p in searching for any target set T in search space Ω, where each map is used in the genetic algorithm conducting the search. Although individual members of the set may induce a probability of success greater than or lower than p, the average probability of success over all the maps is p.

7.4.2 Lemma

Lemma 2. *There exists an infinite number of unbiased fitness map sets.*

Proof. Take an arbitrary unbiased fitness map set, which we label Ω_2, consisting of some number N maps. Such a set is guaranteed to exist, since we take our genetic algorithm to be of type G_a, and can make Ω_2 equal the set of flat fitness maps $F = \{f_i \mid f_i(j) = i \text{ for all } j \in \Omega, 1 \leq i \leq N\}$. Such a set, by definition, has an average probability of success equal to p, since each element of the set does.

We then create a set Θ of an additional N flat fitness maps, such that $\Theta = \{f_i \mid f_i(j) = i \text{ for all } j \in \Omega, N+1 \leq i \leq 2N\}, |\Theta| = |\Omega_2|$ and $\Omega_2' = \Omega_2 \cup \Theta$.

Let μ_i denote the probability of success induced by the ith member of a set. Therefore, the average probability of success for Ω_2' is

$$\begin{aligned}
\frac{\sum_{\omega \in \Omega_2'} \mu_\omega}{|\Omega_2'|} &= \frac{\sum_{i \in \Omega_2} \mu_i + \sum_{j \in \Theta} \mu_j}{2|\Omega_2|} \\
&= \frac{p|\Omega_2| + p|\Omega_2|}{2|\Omega_2|} \\
&= \frac{p|\Omega_2|}{|\Omega_2|} \\
&= p
\end{aligned} \tag{7.17}$$

Since each element of Θ is distinct from elements in Ω_2 (formally $\Omega_2 \cap \Theta = \emptyset$), we see that Ω_2' is distinct from Ω_2 and is an unbiased fitness map set. Since we have done this for an arbitrary unbiased fitness map set, we can do this for any unbiased fitness map set, including Ω_2'. Therefore, the number of unbiased fitness map sets of increasing size is unbound, proving our lemma. □

7.5 Baseline Supersets for Measuring Active Information Costs

7.5.1 Definitions

Definition 7.5.1 (Reduction Information Cost I_r). Let Ω_2 and Ω_2' be sets[1] such that $\Omega_2 \subseteq \Omega_2'$. Then we define the *reduction information cost* $I_r = -\log_2\left(\frac{|\Omega_2|}{|\Omega_2'|}\right)$, where $|\Omega_2|$ denotes the cardinality of Ω_2. Previous work (Dembski and Marks, 2009b) has defined this as the *Brillouin active information* and presented a special case of this concept as the *endogenous information of the search-for-the-search* (Dembski and Marks, 2009a).

7.5.2 Minimum Reduction Information Cost

Lemma 3. *A reduction from the smallest superset Ω_2' of Ω_2 to the set Ω_2 incurs the minimum reduction information cost.*

Proof. We assume, for the sake of later contradiction, that reducing to Ω_2 from a larger superset Ω_2'' incurs a smaller reduction information cost. Therefore,

$$|\Omega_2''| > |\Omega_2'| \tag{7.18}$$

and

$$I_{r_{\Omega_2''}} < I_{r_{\Omega_2'}} \tag{7.19}$$

[1]The subscript 2 is used here to differentiate the original search space Ω from the space of fitness maps over elements in Ω, namely Ω_2. The proofs that follow are made with reductions of the space of fitness maps in mind, hence reference to Ω_2, though they apply to all finite sets of elements.

Substituting the definition of reduction information cost into (7.19), we get

$$
\begin{aligned}
-\log_2\left(\frac{|\Omega_2|}{|\Omega_2''|}\right) &< -\log_2\left(\frac{|\Omega_2|}{|\Omega_2'|}\right) \\
\frac{|\Omega_2|}{|\Omega_2''|} &> \frac{|\Omega_2|}{|\Omega_2'|} \\
\frac{|\Omega_2'|}{|\Omega_2''|} &> 1 \\
|\Omega_2'| &> |\Omega_2''| \qquad (7.20)
\end{aligned}
$$

contradicting our premise (7.18), thus completing our proof. □

7.5.3 Non-Uniqueness of Smallest Unbiased Superset

Lemma 4. *If Ω_2 is a set of fitness maps, the smallest unbiased superset containing Ω_2 is not necessarily unique.*

Proof. Take the special case where Ω_2', the smallest unbiased superset of Ω_2, contains exactly one flat fitness map not in Ω_2. As proven previously, every flat fitness map induces a probability of success equal to p, where p is the probability of success for a null, uniform random sampling search on search space Ω. Since every flat fitness map induces an equal probability of success, take the flat fitness map f_c and replace it with flat fitness map f_{c+1}. This map cannot already be in Ω_2', since Ω_2' has only one flat fitness map. Label this new superset Ω_2''. This set has the same average probability of success as Ω_2', since we replaced one flat fitness map with another that induces the same probability of success. Furthermore, $|\Omega_2'| = |\Omega_2''|$, and $\Omega_2' \neq \Omega_2''$, proving our lemma. □

7.5.4 Non Performance Improving Reductions Exist

Lemma 5. *All reductions from Ω_2' to Ω_2, where $\Omega_2 \subset \Omega_2'$, incur a positive reduction information cost and not all reductions from Ω_2' to Ω_2 result in positive active information.*

Proof. Consider any two sets of fitness maps Ω_2 and Ω_2', where $\Omega_2 \subset \Omega_2'$. Therefore, the reduction information cost $I_r = -\log\left(\frac{|\Omega_2|}{|\Omega_2'|}\right)$, which is greater than zero, since $\frac{|\Omega_2|}{|\Omega_2'|}$ is less than one (by definition). This proves the first part of our lemma.

Additionally, assume that both sets are unbiased. By the definition of unbiased sets of fitness maps, the probability of success induced by both Ω_2 and Ω_2' is p, so that the active information I_+ produced by the reduction is

$$\begin{aligned} I_+ &= \log_2\left(\frac{p}{p}\right) \\ &= 0 \end{aligned} \tag{7.21}$$

completing our proof. □

7.5.5 Discussion

When measuring active information or the endogenous information of a search-for-a-search, the establishment of baselines is important. In prior work (Dembski and Marks, 2009a), a baseline set Ω_2 of search algorithms was assumed to be "the finite space of all search algorithms on the search space Ω." The proofs here show that it is unnecessary to use the set of all possible search algorithms (or in our case, the set of all fitness maps) as a baseline, which would incur the maximum reduction information cost. By choosing instead the smallest possible unbiased superset, we are guaranteed to incur the minimum reduction information cost. This keeps one from arbitrarily inflating information costs, but also prevents one from ignoring the information costs incurred by beginning with a biased superset of fitness maps as a baseline.

7.6 Probability of a Good Target

Lemma 6. *Any target set T of search space Ω, $T \subseteq \Omega$, providing b bits of active information must have a size $|T| \leq \frac{|\Omega|}{2^b}$.*

Proof. This follows from the definition of active information. Since $p = \frac{|T|}{|\Omega|}$ and $0 \leq q \leq 1$, we have

$$\begin{aligned} b &= I_+ \\ &= -\log_2\left(\frac{p}{q}\right) \\ &= \log_2\left(\frac{q}{\frac{|T|}{|\Omega|}}\right) \\ &= \log_2\left(\frac{q|\Omega|}{|T|}\right) \\ |T| &= \frac{q|\Omega|}{2^b} \\ &\leq \frac{|\Omega|}{2^b} \end{aligned} \tag{7.22}$$

□

Lemma 7. $\sum_{k=0}^{\frac{n}{2^b}} \binom{n}{k} \geq 2\sum_{k=0}^{\frac{n}{2^{b+1}}} \binom{n}{k}$

Proof. We begin with the inequality

$$\sum_{k=\frac{n}{2^{b+1}}+1}^{\frac{n}{2^b}} \binom{n}{k} \geq \sum_{k=0}^{\frac{n}{2^{b+1}}} \binom{n}{k}.$$

Notice that since both summations have the same number of terms and the binomial coefficient is nondecreasing for $b \geq 0$, the left hand side has larger terms and the inequality holds. We continue adding terms to both sides of the inequality,

$$\begin{aligned} \sum_{k=0}^{\frac{n}{2^{b+1}}} \binom{n}{k} + \sum_{k=\frac{n}{2^{b+1}}+1}^{\frac{n}{2^b}} \binom{n}{k} &\geq \sum_{k=0}^{\frac{n}{2^{b+1}}} \binom{n}{k} + \sum_{k=0}^{\frac{n}{2^{b+1}}} \binom{n}{k} \\ \sum_{k=0}^{\frac{n}{2^b}} \binom{n}{k} &\geq 2\sum_{k=0}^{\frac{n}{2^{b+1}}} \binom{n}{k}, \end{aligned} \tag{7.23}$$

thus completing our proof. □

Lemma 8.

$$\sum_{k=0}^{\frac{n}{2^b}} \binom{n}{k} \leq 2^{n-b} \tag{7.24}$$

Proof.

$$\begin{aligned}
\sum_{k=0}^{n}\binom{n}{k} &= 2^0\sum_{k=0}^{\frac{n}{2^0}}\binom{n}{k} \\
&\geq 2^0 2\sum_{k=0}^{\frac{n}{2^1}}\binom{n}{k} && \text{(By Lemma 7)} \\
&= 2^1\sum_{k=0}^{\frac{n}{2^1}}\binom{n}{k} \\
&\geq 2^2\sum_{k=0}^{\frac{n}{2^2}}\binom{n}{k} && \text{(By Lemma 7)} \\
&\vdots \\
&\geq 2^b\sum_{k=0}^{\frac{n}{2^b}}\binom{n}{k}
\end{aligned}$$

Therefore

$$\begin{aligned}
\sum_{k=0}^{n}\binom{n}{k} &\geq 2^b\sum_{k=0}^{\frac{n}{2^b}}\binom{n}{k} \\
2^n &\geq 2^b\sum_{k=0}^{\frac{n}{2^b}}\binom{n}{k} \\
\frac{2^n}{2^b} &\geq \sum_{k=0}^{\frac{n}{2^b}}\binom{n}{k} \\
2^{n-b} &\geq \sum_{k=0}^{\frac{n}{2^b}}\binom{n}{k} && (7.25)
\end{aligned}$$

□

Theorem 3. *For $b \geq 0$, The number of target sets providing b or more bits of active information in a set S_2 of all possible target sets is less than or equal to $\frac{|S_2|}{2^b}$. The null search probability of finding such a target set in S_2 is, therefore, less than or equal to 2^{-b}.*

Proof. Since there are $2^{|\Omega|}$ subsets possible over Ω, $|S_2| = 2^{|\Omega|}$. Let τ denote the set of target sets in S_2 that provide b or more bits of active information. Formally

$$\tau = \{T \in S_2 \mid I_+ \geq b\} \tag{7.26}$$

By Lemma 6, we have

$$\tau = \left\{T \in S_2 \mid |T| \leq \frac{|\Omega|}{2^b}\right\} \tag{7.27}$$

This definition holds, since any target set providing greater than b bits of active information must also have a size less than or equal to $\frac{|\Omega|}{2^b}$.

Since there are exactly $\sum_{k=0}^{\frac{|\Omega|}{2^b}} \binom{|\Omega|}{k}$ unique subsets of Ω with $\frac{|\Omega|}{2^b}$ or less elements, by applying Lemma 8 we find

$$\begin{aligned} \frac{|\tau|}{|S_2|} &= \frac{\sum_{k=0}^{\frac{|\Omega|}{2^b}} \binom{|\Omega|}{k}}{2^{|\Omega|}} \\ &\leq \frac{2^{|\Omega|-b}}{2^{|\Omega|}} \\ &= \frac{1}{2^b} \end{aligned} \tag{7.28}$$

Rearranging

$$|\tau| \leq \frac{|S_2|}{2^b} \tag{7.29}$$

□

APPENDICES

APPENDIX A

Simple Ordinal Communication Channel

A.1 Overview

Given a list of integers, it is possible to encode messages based on the ordering of the integers in the list. One can consider a Latin alphabet message to be a base 27 number (26 letters plus a space) and define an ordering over lists of integers, such that each permutation of a list corresponds to precisely one Latin alphabet message (base 27 number). The conversion process is to convert the message to a number, then use that number to select the appropriate list of integers. To decode, we calculate a number from the ordering of items in the list, then convert that base 10 number into a base 27 number, which serves as the plaintext message.

In this way, one can see that a set of rankings, such as those provided by fitness functions, can function as a simple communication channel. For example, by ranking students in some order, one can effectively transmit a short message to another party in a steganographic manner.

A.2 Encoding Algorithm

Let x be a finite sequence drawn from an alphabet consisting of the 26 Latin alphabet letters and a space symbol. The encode algorithm consists of the following two steps:

(1) Converting x into a base 10 number s

(2) Converting s to an ordered list of integers

1.2.1 Converting to Base 10

We convert x from a base 27 number to a base 10 number in the standard manner. Let $x = x_0x_1 \dots x_m$ and let x_i denote the ith letter of x, where $0 \leq i < m$.

Let

$$alpha = [\text{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, \_}]$$

where '_' denotes the space character. We calculate s, the base 10 representation of x as

$$s = \sum_{i=0}^{m-1} 27^i \cdot \text{index}(x_i, alpha)$$

where index() is a function that returns the index of a letter in *alpha*, beginning with 'a' at index 0.

1.2.2 Converting s to a List of Items

The next step in the encoding algorithm is to convert s into a list of n integers, where n is the smallest number such that $n! \geq s$. We define a list r, containing integers from 0 to $n-1$ in ascending order.

We then traverse a tree which we use to decide which item from r to select for the next position of the output list. The number of branches at any level of the tree is the number of remaining items in r, and descending a level in the tree represents selecting and removing an item from r. Let d denote the zero-based index of an item in r that we choose as the next item to remove and place in the output list. We calculate d as

$$d = \left\lfloor \frac{s}{(|r|-1)!} \right\rfloor$$

The sum s is then updated as follows

$$s = s - d \cdot (|r|-1)!$$

The procedure is repeated until all items are removed from r and placed in the output list.

A.3 Decoding Algorithm

The decode algorithm is similar to the encode algorithm. We compute the plaintext from a list of integers by the following two steps:

(1) Converting the ordered list of integers into a base 10 number s

(2) Converting s to a base 27 number, i.e. the plaintext message x

1.3.1 Converting List of Items to Base 10

The first step of decoding is to transform the ordered list of integers, q, back to a base 10 positive integer. To do so, we first create another ordered list r, which is an ascending sorted version of the list q. We iterate over the items in q, beginning at position 0, and find the index of the integer at that position in list r. We then multiply that index by $(|r|-1)!$ and add this to our running total. Lastly, we remove the item from r, and continue until we have iterated over all items in list q.

This forms a greedy method of recreating the number s from the list of integers q. At each step, we have a remaining list of items r, which has indices ranging from 0 to $|r|-1$ available. During encoding, the index number chosen is the maximum such that this number multiplied by $(|r|-1)!$ is less than or equal to s. In decoding, we use the number chosen during the encoding step to tell us how many copies of $(|r|-1)!$ we "packed" into s at that step, being the maximum possible, and so we calculate this product and add it to s. Repeating this step for all positions of q allows us to reconstruct the integer s.

1.3.2 Converting from Base 10 to Base 27

The decoding algorithm converts s from a base 10 integer to the base 27 plaintext message x in the standard manner, by repeatedly dividing by the conversion base and using the remainder as the digit to output at that step.

A.4 Usage

To use the encoder code supplied below, simply run encode.py from the command line using the python interpreter (version 2.6.*), and feed into standard input the phrase you'd like encoded.

```
\> python encode.py
input: hello world
output: [1, 17, 13, 5, 4, 0, 3, 12, 8, 15, 14, 11, 16, 7, 9, 10, 2, 6]
```

To decode, run the decode.py script and pass in the list of integers to standard input, in the same format as given by the encode method, which will produce the plaintext message.

```
\> python decode.py
input: [1, 17, 13, 5, 4, 0, 3, 12, 8, 15, 14, 11, 16, 7, 9, 10, 2, 6]
output: hello world
```

A.5 Code

1.5.1 encode.py

```
import sys


def next_factorial(s):
    i = 1.0
    while s >= 1:
        i += 1
        s = s / i
    return int(i)


def facs_table(n):
    facs = [1,]
    for i in range(n - 1):
        facs.append(facs[-1] * len(facs))
    return facs
```

```
def main(content):
    x = content.strip()
    alpha = "abcdefghijklmnopqrstuvwxyz "[:]
    s = sum([(len(alpha)**i) * alpha.index(letter) \
            for i, letter in enumerate(x[:])])
    n = next_factorial(s)
    items = [i for i in range(n)]
    out = []
    facs = facs_table(n)
    for i in range(n):
        d = int(s / facs[-(i + 1)])
        s -= (d * facs[-(i + 1)])
        out.append(items[d])
        del items[d]
    print(out)


if __name__ == "__main__":
    main(sys.stdin.read())
```

1.5.2 decode.py

The following code requires installation of the `mpmath` python module.

```
import sys
from mpmath import mp, mpf
mp.dps = 1000


def facs_table(n):
    facs = [1,]
    for i in range(n - 1):
```

```
        facs.append(facs[-1] * len(facs))
    return facs


def main(content):
    q = [int(item) for item in \
         content.strip(" []\n ").split(",")]
    alpha = "abcdefghijklmnopqrstuvwxyz "[:]
    radix = len(alpha)
    out = ""
    r = q[:]
    r.sort()
    facs = facs_table(len(q))
    s = 0
    for i, item in enumerate(q):
        s += r.index(item) * facs[-(i + 1)]
        r.remove(item)
    while s > 0:
        part = int(s % mpf(radix))
        out += alpha[part]
        s = int(s / mpf(radix))
    print(out)


if __name__ == "__main__":
    main(sys.stdin.read())
```

APPENDIX B

Implementation of a Near Optimal Black-Box Search Algorithm

B.1 Algorithm

Having calculated in Chapter 3 Section 3.3 the upper bound for active information produced by a black-box search algorithm during a single search, we now consider an example of how a near optimal search algorithm can be implemented.

We consider a fitness map f over the elements of the finite search space Ω containing a target T. The set T consists of a single element of Ω, which we label t_1. The fitness map is then calculated according to the following function

$$f(x) = t_1 \oplus x \tag{B.1}$$

where $\oplus$ denotes the exclusive-or (XOR) operation over the bit strings t_1 and x.

Given this fitness map, the algorithm works as follows:

(1) The algorithm queries an arbitrary point ω in Ω. If the element queried is the target t_1, the algorithm terminates. If not, the element is evaluated according to the fitness map, and a binary string y is returned that represents the fitness value of that element.

(2) The binary string y is applied as an XOR mask on the element ω, producing the target string.

(3) All probability mass is placed on the target value computed in the previous step, forming distribution $\pi_{I(t)}$.

(4) A new point is selected from Ω according to the distribution $\pi_{I(t)}$, locating t_1 and terminating the search.

Using this algorithm and fitness map, a target in Ω can be located in a maximum of two queries.

B.2 Discussion

This implementation assumes that target information is available beforehand and that the algorithm is able to perfectly exploit this information. In many real-world situations neither assumption holds, so the above algorithm serves as a proof-of-concept rather than a practical strategy for solving search problems.

Interestingly, the same fitness map could be used by a standard genetic algorithm by counting the number of ones in fitness value y and using this as a Hamming distance. Hamming fitness functions are sometimes used in model genetic algorithms (Schneider, 2000) and can be easily computed from a target sequence. By converting the fitness map f into a Hamming fitness map some information is lost, since we then only know the number of differences and not where they occur. Therefore, there is a corresponding decrease in performance compared to the near optimal search. Several methods exist for extracting target information from a Hamming fitness map, some more efficient than others (Ewert et al., 2010).

BIBLIOGRAPHY

Abu-Mostafa, Y., and J. St Jacques. "Information capacity of the Hopfield model." *Information Theory, IEEE Transactions on* 31, 4: (1985) 461–464.

Altenberg, L. "The Schema Theorem and Price's Theorem." *Foundations of genetic algorithms 3* 23.

———. "NK fitness landscapes." *Handbook of evolutionary computation* 2.

Bäck, T., and H.P. Schwefel. "An overview of evolutionary algorithms for parameter optimization." *Evolutionary computation* a, 1: (1993) 1–23.

De Jong, K.A. "Analysis of the behavior of a class of genetic adaptive systems." *Dissertation Abstracts International* 36.

Dembski, W.A., and R.J. Marks. "Bernoulli's principle of insufficient reason and conservation of information in computer search." In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*. 2009a, 2647 –2652.

———. "Conservation of Information in Search: Measuring the Cost of Success." *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 39, 5: (2009b) 1051 –1061.

English, T.M. "Optimization is easy and learning is hard in the typical function." In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*. IEEE, 2000, volume 2, 924–931.

———. "No more lunch: Analysis of sequential search." In *Evolutionary Computation, 2004. CEC2004. Congress on*. IEEE, 2004, volume 1, 227–234.

Ewert, W., G. Montañez, W. Dembski, and R. Marks. "Efficient per query information extraction from a Hamming oracle." In *System Theory (SSST), 2010 42nd Southeastern Symposium on*. IEEE, 2010, 290–297.

Floreano, D., and L. Keller. "Evolution of adaptive behaviour in robots by means of darwinian selection." *PLoS Biol* 8, 1.

Goldberg, D.E. *Genetic algorithms in search, optimization, and machine learning*. Addison-wesley, 1989.

He, Jun, Colin Reeves, Carsten Witt, and Xin Yao. "A note on problem difficulty measures in black-box optimization: Classification, realizations and predictability." *Evol. Comput.* 15, 4: (2007) 435–443.

Holland, J.H. *Adaptation in natural and artificial systems*. The University of Michigan Press, Ann Arbor, 1975.

Jones, Terry, and Stephanie Forrest. "Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms." In *Proceedings of the Sixth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1995, 184–192.

Montañez, G., W. Ewert, W. Dembski, and R. Marks. "A Vivisection of the ev Computer Organism: Identifying Sources of Active Information." *BIO-Complexity* 2010, 0.

Radcliffe, N.J. "Schema processing." *Handbook of Evolutionary Computation* 2.

Reeves, C.R., and J.E. Rowe. *Genetic algorithms: principles and perspectives: a guide to GA theory*. Kluwer Academic Pub, 2002.

Schaffer, Cullen. "A Conservation Law for Generalization Performance." In *ICML'94*. 1994, 259–265.

Schneider, T. D. "Evolution of Biological Information." *Nucleic Acids Res.* 28, 14: (2000) 2794–2799.

Schumacher, C., MD Vose, and LD Whitley. "The no free lunch and problem description length." In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*. Citeseer, 2001, 565–570.

Shannon, C.E. "A mathematical theory of communication." *Bell System Technical Journal* 27, 3: (1948) 379–423.

Sywerda, G. "Uniform crossover in genetic algorithms." In *Proceedings of the third international conference on Genetic algorithms*. Morgan Kaufmann Publishers Inc., 1989, 2–9.

Van Nimwegen, E., J.P. Crutchfield, and M. Mitchell. "Finite populations induce metastability in evolutionary search." *Physics Letters A* 229, 3: (1997) 144–150.

Vose, M.D. *The simple genetic algorithm: foundations and theory*, volume 12. The MIT Press, 1999.

Wolpert, D.H., and W.G. Macready. "No free lunch theorems for optimization." *IEEE Transactions on Evolutionary Computation* 1, 1: (1997) 67–82.