ABSTRACT

Studies of Active Information in Search

Winston Ewert, M.S.

Chairperson: Gregory J. Hamerly, Ph.D.

A search process is an attempt to locate a solution to a problem, such as an optimization problem, where the space is usually too large to exhaustively sample. In order to investigate this idea this work looks a three examples of searches as cases studies. The examples considered are the location of a hidden string using a hamming distance, the encoding of a binary string using a perceptron, and developing programs using nand gates. In all of these cases, it is shown that the search processes work by making use of problem specific information. In addition, the algorithms used to demonstrate these search processes are often relatively inefficient at extracting the information from the available knowledge sources.

Studies of Active Information in Search

by

Winston Ewert, 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

_____

Young-Rae Cho, Ph.D.

_____

Robert Marks, Ph.D.

Accepted by the Graduate School
December 2010

_____

J. Larry Lyon, Ph.D., Dean

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

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREAMBLE

This thesis is a replacement of a previous version. The replacement is necessary because of a number of serious challenges. Some of the contents of the first version were taken from previously published work of the author. As was done in the original version, copyright notices are included at the end of the thesis. Although using previously published work in a thesis is common practice, the document itself did not make clear that this was the case. Additionally, and more seriously, the introduction was constructed by drawing passages from previous papers including some which the author of thesis was not a coauthor. This was clearly an inappropriate usage. The work for this thesis was done in collaboration with others in Dr. Marks' research group. All of content in the thesis including the sections from the introduction were produced by members of the research group including Dr. Marks who was a member of the thesis committee. Within the group, we use tools such as LaTeXand Dropbox which make collaboration very easy. Unfortunately this also made it easy to reuse existing text in an inappropriate manner. This is not an attempt to excuse the content of the original thesis, but rather to explain how the mistake was made. In future all members of the lab, especially the author of this thesis, will be careful about the reuse of collaborative work. This version of the thesis remedies these problems. The introduction has been rewritten by the author and those sections drawn from previous work of the author have been appropriately cited in the thesis body.

CHAPTER ONE

Introduction [1]

It is often the case that we have a large set of potential solutions to a given problem; a search process is required to select the desired solution from this set. Typically, we can directly evaluate the performance of a particular solution. However, this evaluation is typically expensive, limiting the amount of such evaluations that can be performed. The problem of search is to select which solutions to evaluate taking into account all available information including the performance of any previously evaluated solutions.

For any particular search problem, there exists knowledge about the problem before it is attempted. Any detail which is known to be true about the search problem is an instance of knowledge. When this knowledge is represented in a form which can actually help a computer find the solution, such as a computer a algorithm, we term it information. Any search algorithm does not produce information; rather, it makes use of the information which was encoded into it by its programmers. A pioneer in the area of information theory, Leon Brillouin, wrote:

> The [computing] machine does not create any new information, but it performs a very valuable transformation of known information. (Brillouin, 1962)

If it is impossible for the computer to create new information, then any success in a search must derive from information sources provided to the search. Various formalizations of these results have been made both in the area of search and machine learning. (Schaffer, 1994; Wolpert and Macready, 1997; Ho and Pepyne, 2002; English,

[1] The contents of the introduction parallels, but is not drawn verbatim, from the introduction to Dembski, William A., and Robert J. Marks. "Conservation of information in search: measuring the cost of success. Trans. Sys. Man Cyber. Part A 39, 5: (2009b) 1051-1061. This paper provides the foundation for much of this thesis.

1999) In the case of search algorithms, these results imply that all search algorithms which do not take advantage of prior knowledge about a problem will have the same poor performance. As a result, any search algorithm necessarily requires external assistance in order to reliably locate its target. Consequently, any search algorithm which succeeds with high probably must either be facing an easy problem or be making use of problem specific information.

What are the implications of this conservation of information? These results show that no search algorithm has an intrinsic advantage over any other search algorithm. On average, they will all perform poorly. This would seem to be at odds with the successful widespread application of various search techniques (Kennedy, 2001; Tomasz, 2006; Reed and Marks II, 1999). In fact, techniques such as swarm optimization or genetic algorithms are commonly used to solve an impressive array of problems. However, these methods still allow the programmer to inject information into the search algorithm such as by the careful design of a penalty or fitness function. In fact, some such programmers have called themselves "penalty function artists" (Healy, 2001).

Perhaps the most typical source of information in a search problem is an oracle (Dembski and Marks, 2009b; Jai et al., 1990, 1991; Jensen et al., 1999; Oh et al., 1991). An oracle is any process which provides answers to questions posed by the search process. Often, the question takes from of evaluating the performance of a particular solution. We assume that this evaluation is considerably more expensive then anything in the search algorithm, and thus we can measure the computational cost of a search algorithm by the number of queries that it performs.

1.1 Measurement Method

In order to gain insight into the information extracted from oracles, we seek to measure the information they contribute. This is accomplished by *active information* (Dembski and Marks, 2010b, 2009b, 2010a). We measure information by taking

logarithms of probabilities, following the practice of information theory (Shannon, 1948).

Consider the *information* (Dembski and Marks, 2010b, 2009b, 2010a) required in order for a search succeed. In other words, how difficult is the search? What probability does it have of success?

$$-\log_2 p \text{ bits} \tag{1.1}$$

where p denotes the probability of the search in question succeeding.

Information is a logarithmic measurement; consequently, it must be measured relative to another quantity. This means that we measure the information in one search by comparing it to another search. Typically, we use an unassisted search as a baseline. An unassisted search is one which due to lack of prior knowledge about the problem can only guess at the correct solution. In such a search, we call the information quantity defined above *endogenous information*. In contrast, the search we are interested in is the assisted search, which takes into account some prior knowledge about the problem it is facing. Here we term the same quantity, *exogenous information*. The *active information* is defined as being the difference between the exogenous information and the endogenous information or the increase in information produced by the assistance present in the assisted search. This quantity enables us to measure the value of assistance provided to searches.

Consider the case where a large quantity of Styrofoam cups are placed opening downwards on a table. Some of these cups conceal prizes whereas the majority have nothing underneath them. The implication of conservation of information results is that in the absence of additional information no strategy exists to help select the correct cup. If there are 1024 cups and only a single prize, the probability of selecting the correct cup is $\frac{1}{1024}$. The endogenous information of this search is $-\log_2 \frac{1}{1024} = 10$ bits.

However, if we are able to eliminate some of the cups from consideration, we can increase our chance of success. After the elimination of half of the cups, the probability of success is $\frac{1}{512}$ and the exogenous information is 9 bits. This gives an active information of 1 bit. The active information gives us an indication of how helpful that additional information was.

However, if the prize really was in half of the cups eliminated, our chance of success is now zero, leading to negative infinity active information. Far from being helpful, such "assistance" actually removed the chance at success.

1.2 Queries

A search typically makes use of queries as the primary information source. As such, we desire to measure the information derived from each individual query. The importance of per-query information derives from the fact that any oracle will have resource constraints. That is, there is a limit to the number of questions that the search algorithm can ask the oracle. If there is no limitation, the search can simply ask every possible question making the best strategy trivial.

We denote the number of performed queries as Q and have $\acute{Q}$ as the upper limit on the number of queries performed. From this we can define the active information per query.

$$I_{\oplus} := \mathbb{E}\left[\frac{I_+}{Q}\right] \tag{1.2}$$

In trivial cases, it is possible to analytically determine this formula. However, we will approximate the formula either using another analytical technique or through Monte Carlo simulations.

CHAPTER TWO

Hamming Oracle [1]

2.1 Introduction

Oracles can be used with different degrees of efficiency. In the most simple of examples using a *needle in a haystack* oracle, blind sampling without replacement outperforms blind sampling with replacement in terms of average query count. In many cases, evaluation of oracle query efficiency can only be answered through extensive Monte Carlo simulation. An exception is the Hamming oracle which is well suited for analytical study. We have a target of length L using an alphabet of N, *e.g*

```
A*PHRASE*HIDDEN*AMONG*MANY*PHRASES
```

has $L = 34$ letters from an alphabet of $N = 27$ (26 letters and a space). When a sequence of letters is presented to a Hamming oracle, the oracle responds with the Hamming distance equal to the number of letter mismatches in the sequence.

There are numerous ways in which the Hamming oracle can (and has) been used to find unknown phrases (Dembski and Marks, 2009b; Mackay, 2004; Schneider, 2000). In terms of query count, some are more efficient than others. How is this simple oracle best interrogated using the currency of queries? We analyze commonly used Markov based evolutionary algorithms and demonstrate, by comparison, that the resulting stochastic hill climbing searches use oracles better than blind search. The *frequency of occurrence (FOO) Hamming oracle algorithm* (FOOHOA) is a deterministic algorithm that performs significantly better than stochastic hill climbing. Unlike a Markov

[1] A previous version of this chapter was published as: Winston Ewert, George Montañez, William A. Dembski, Robert J. Marks II, "Efficient Per Query Information Extraction from a Hamming Oracle," Proceedings of the 42nd Meeting of the Southeastern Symposium on System Theory, IEEE, University of Texas at Tyler, March 7-9, 2010, pp.290-297. The copyright notice is in Appendix A. Much of this chapter, including figures, is taken verbatim from this work.

approach which uses only its current state to determine its next step, information extracted during the entire history of the search is used to determine the next query.

Is it possible to know, in general, when a search built around a fixed oracle is optimal? Alternately, given an oracle, what is the maximum amount of information that can be extracted on a per query basis? A *search for the search* (S4S) (Dembski and Marks, 2010b) attempts to find either a search algorithm that exceeds specified performance criteria or, in the extreme, to find the algorithm with the globally best performance. Given a Hamming oracle, we present a search over all possible search trees that use the Hamming oracle to find the tree with minimum average depth. The corresponding search generates the greatest amount of active information per query of all such tree searches. The S4S, however, is so computationally intense, optimal trees can only be found for short messages using small alphabets.

Study of efficient extraction of information from the Hamming oracle is instructive in understanding the difficulty and obstacles of efficient information extraction from other oracles where analytic tractability is not as friendly.

2.2 Markov Models of Evolutionary Search

Markov models are useful in describing certain evolutionary searches (Bäck, 1996; Spears, 1995; Stewart, 2009). Before analysis, a brief background on Markov processes is required.

2.2.1 Markov Processes With an Absorbing State

We are searching for a target message of length L. In general, an $(L+1)\times(L+1)$ Markov matrix, $\mathbf{P}$, has elements

$$(\mathbf{P})_{k,\lambda} = p_{k,\lambda} \tag{2.1}$$

where $p_{k,\lambda}$ is the probability of going to state k given we are in state λ. When all L letters of a sequence have been defined, the process is stopped at the final *absorbing state* from which there is no escape. The absorbing state is the target message and,

when achieved, the search is announced as a success. The corresponding Markov matrix with absorbing state L is

$$\mathbf{P} = \left[\begin{array}{cccc|c} p_{0,0} & p_{0,1} & \cdots & p_{0,L-1} & 0 \\ p_{1,0} & p_{1,1} & \cdots & p_{1,L-1} & 0 \\ p_{2,0} & p_{2,1} & \cdots & p_{2,L-3} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ p_{L-2,0} & p_{L-2,1} & \cdots & p_{L-2,L-1} & 0 \\ p_{L-1,0} & p_{L-1,1} & \cdots & p_{L-1,L-1} & 0 \\ \hline p_{L,0} & p_{L,1} & \cdots & p_{L,L-1} & 1 \end{array}\right]$$

$$= \left[\begin{array}{c|c} \mathbf{S} & \vec{0} \\ \hline \vec{q}^{\,T} & 1 \end{array}\right]$$

where, for clarity, we have partitioned the $\mathbf{P}$ matrix, $\mathbf{S}$ is an $L \times L$ matrix, and

$$\vec{q} = [p_{L,0} \;\; p_{L,1} \;\; \cdots \;\; p_{L,L-1}]^T.$$

Then the iteration $\vec{\pi}(n+1) = \mathbf{P}\vec{\pi}(n)$ has the solution

$$\vec{\pi}(n) = \mathbf{P}^n \vec{\pi}(0) = \left[\begin{array}{c|c} \mathbf{S}^n & \vec{0} \\ \hline \vec{q}_n^{\,T} & 1 \end{array}\right] \vec{\pi}(0). \tag{2.2}$$

where $\vec{q}_n^{\,T}$ is a vector the exact values of which are not important. The probability of going from state λ to k in the nth iteration is $(\mathbf{S}^n)_{k,\lambda}$. Summing all of these probabilities for all (k, λ) gives the geometric series

$$\sum_{n=0}^{\infty} \mathbf{S}^n = [\mathbf{I} - \mathbf{S}]^{-1}. \tag{2.3}$$

The matrix[2] $\mathbf{F} := [\mathbf{I} - \mathbf{S}]^{-1}$, dubbed the *functional matrix*, has elements

$$(\mathbf{F})_{k,\lambda} = \begin{cases} \text{expected number of the} \\ \text{times state } \lambda \text{ has gone to} \\ \text{state } k. \end{cases} \tag{2.4}$$

Define $\vec{N} = \vec{1}^T\mathbf{F}$ where $\vec{1}$ is a vector of 1's. Then $(\vec{N})_k$ is the number of steps to absorption assuming an initialization of state k.

- If we begin with $\lambda = 0$ correct characters, then the initialization, $\vec{\pi}(0)$, has a one in the first place and is otherwise zero. In this case, $G = (\vec{N})_0$ where G is the expected number of generations to success.
- A better initialization would be to start with a $\vec{\pi}(0)$ vector whose probabilities reflect selection from a randomly chosen initialization. Let G denote the expected number of generations. Then

$$G = \vec{N}^T\vec{\pi}(0) \tag{2.5}$$

2.2.2 Information Measures in Markov Searches

The endogenous information of a search for L letters using an N character alphabet is

$$I_\Omega = L \log_2 N.$$

If each iteration requires K queries (corresponding to K children) the expected number of queries to do a perfect search is $Q = KG$ where G is given by (2.5). So the active information per query is

$$I_\oplus = \frac{L \log_2 N}{KG}. \tag{2.6}$$

For the purposes of this study, we are not following Equation 1.2. We use $\frac{I_\Omega}{\mathbb{E}[Q]}$ rather then $\mathbb{E}[\frac{I_\Omega}{Q}]$. However as a result of Jensen's inequality (Cover, 1991), $\mathbb{E}[I_\Omega/Q] \geq$

[2] In some cases, the matrix $\mathbf{I} - \mathbf{S}$ may be ill-conditioned (Marks II, 2009). In our analysis, we only consider inversion of matrices with condition numbers not exceeding 10^5.

$1/\mathbb{E}[Q]$. Since we are consistent in the use of this form, the comparison between algorithms remains valid.

For all of the Markov matrices considered here, we will use

$$\lambda = \text{ the number of correct letters}$$

as the state. We will start with a randomly selected string which could be in any of the possible states. To determine the probability of starting in a particular state, we need to calculate the probability of a random string having that many letters in common with the target string. We can calculate the vector of starting positions to be used in the initialization as

$$\vec{\pi}(0) = \begin{bmatrix} \Pr[\lambda = 0] \\ \\ \Pr[\lambda = 1] \\ \\ \Pr[\lambda = 2] \\ \vdots \\ \Pr[\lambda = k] \\ \vdots \\ \Pr[\lambda = L-1] \end{bmatrix} = \begin{bmatrix} q^L \\ \\ \binom{L}{1} q^{L-1} p \\ \\ \binom{L}{2} q^{L-2} p^2 \\ \vdots \\ \binom{L}{k} q^{L-k} p^k \\ \vdots \\ \binom{L}{L-1} q p^{L-1} \end{bmatrix}$$

where $p = 1/N$, $q = 1 - p$, and the binomial coefficient is $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. We will use this initialization for all of the algorithms to extract the expected number of queries or generations before absorption.

Once we have determined the matrix associated with particular algorithm, we can use (2.5) to calculate the expected number of iterations of the algorithm and thus determine the active information per query extracted through the algorithm.

2.2.3 K Children Each With a Single Mutation

We now apply the Markov model to the analysis of four different evolutionary search strategies. Unless otherwise indicated, all probabilities in this section are assumed to be nonzero only on the interval $0 \leq \lambda \leq L$.

In the single mutation scenario, only one of the L characters is changed in every child of the parent. This is done by selecting a random position in the sequence and changing that position to a random character drawn from the alphabet. We first consider the case where there is a single child and then when K children are generated and the best fitness among the children is kept for the next parent.

2.2.3.1 *Mutation of One Character per String.* **For One Child.** We define the following events

- B means the score is better,
- S means the score is the same,
- W means the score is worse,
- g (for good) means the randomly chosen position already matches the target, and
- $b = \bar{g}$ (for bad) means it doesn't.

Using the *theorem of total probability*, we evaluate three cases.

(1) **Worse.** The probability of a worse score is

$$\begin{aligned} \Pr[W] &= \Pr[W|g]\Pr[g] + \Pr[W|b]\Pr[b] \\ &= \frac{N-1}{N} \times \frac{\lambda}{L} + 0 \times \frac{L-\lambda}{L} = \frac{(N-1)\lambda}{NL}. \end{aligned} \tag{2.7}$$

(2) **Better.** The probability of a better score is

$$\begin{aligned} \Pr[B] &= \Pr[B|g]\Pr[g] + \Pr[B|b]\Pr[b] \\ &= 0 \times \frac{\lambda}{L} + \frac{1}{N} \times \frac{L-\lambda}{L} = \frac{L-\lambda}{NL}. \end{aligned} \tag{2.8}$$

(3) **Same.** The probability of the same score is

$$\begin{aligned}\Pr[S] &= \Pr[S|g]\Pr[g] + \Pr[S|b]\Pr[b] \\ &= \frac{1}{N} \times \frac{\lambda}{L} + \frac{N-1}{N} \times \frac{L-\lambda}{L} \\ &= \frac{\lambda + (N-1)(L-\lambda)}{NL}. \end{aligned} \tag{2.9}$$

Note that, as expected, $\Pr[B] + \Pr[S] + \Pr[W] = 1$.

2.2.3.2 *For K Children.* We now consider K offspring. The mutation result of one child is independent of another. The best child is chosen to be the parent of the next generation. When there is a tie, one is chosen at random. We will assume the parent has λ correct characters and analyze whether we do better $(\lambda + 1)$, worse $(\lambda - 1)$, or the same (λ). Let's consider the same three cases. [3]

(1) **Worse.** The only way to get a worse score is for all the children to have a worse score. Thus, using (2.7),

$$\Pr[\lambda - 1|\lambda] = (\Pr[W])^K = \left(\frac{(N-1)\lambda}{NL}\right)^K. \tag{2.10}$$

(2) **Better.** The probability of at least one of the children having a better score than the parent is the complement of none of the children having a better score. Thus, using (2.8),

$$\begin{aligned}\Pr[\lambda + 1|\lambda] &= 1 - \left(\Pr[\bar{B}]\right)^K \\ &= 1 - (1 - \Pr[B])^K = 1 - \left(1 - \frac{L-\lambda}{NL}\right)^K \end{aligned} \tag{2.11}$$

[3] As $K \to \infty$, we expect to always have a better result. This is confirmed by the asymptotic values of probabilities in (2.10), (2.11) and (2.12).

- **Worse.** From (2.10), $\lim_{K\to\infty} \Pr[\lambda - 1|\lambda] = 0$.
- **Better.** From (2.11), $\lim_{K\to\infty} \Pr[\lambda + 1|\lambda] = 1 - \lim_{K\to\infty} \left[1 - (L-\lambda)/(NL)\right]^K = 1$.
- **Same.** Lastly, from (2.12), $\lim_{K\to\infty} \Pr[\lambda|\lambda] = \lim_{K\to\infty} \left(1 - (L-l)/(NL)\right)^K - \lim_{K\to\infty} \left[((N-1)l)/(NL)\right]^K = 0$

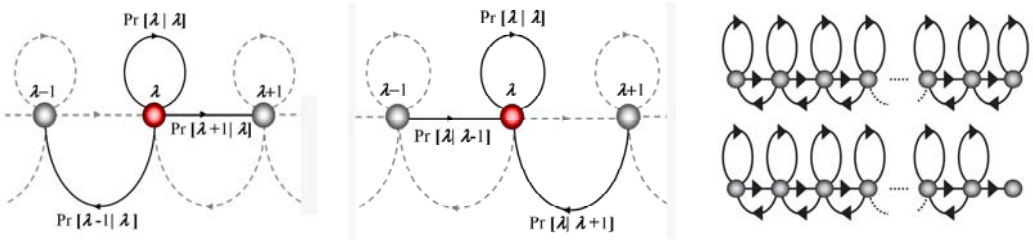

Figure 2.1: Markov Chains for the Single Mutation Per Child Case
LEFT: The search problem as a Markov chain. The l node is shown as a source node.
CENTER: An equivalent representation of the Markov model with l as a sink node.
RIGHT: Non absorbing (top) and absorbing (bottom) cases.

(3) **Same.** Since

$$\Pr[\lambda|\lambda] = 1 - (\Pr[\lambda - 1|\lambda] + \Pr[\lambda + 1|\lambda]),$$

we conclude from (2.10) and (2.11) that

$$\Pr[\lambda|\lambda] = \left(1 - \frac{L-\lambda}{NL}\right)^K - \left(\frac{(N-1)\lambda}{NL}\right)^K. \tag{2.12}$$

With the state λ as a source node, this analysis can be interpreted using the Markov chain shown on the left in Figure 2.1 with transitional probabilities given by (2.10), (2.11) and (2.12). As shown in the middle of Figure 2.1, the chain can be equivalently viewed with λ as a sink node. For the chain with an absorbing state shown on the bottom right in Figure 2.1, there are two special cases to consider.

(1) For $\lambda = 0$, we have $\Pr[\lambda|\lambda - 1] = 0$.

(2) For $\lambda = L$, we have $\Pr[\lambda|\lambda + 1] = 0$.

Otherwise we have the following.

(1) **Coming from state $\lambda + 1$.** Using (2.10), we obtain

$$\Pr[\lambda|\lambda + 1] = ((N-1)(\lambda+1)/NL)^K$$

(2) **Coming from state $\lambda - 1$.** Likewise, with $\lambda \to \lambda - 1$, (2.11) gives

$$\Pr[\lambda|\lambda - 1] = 1 - \left(\frac{(N-1)L + (\lambda - 1)}{NL}\right)^K$$

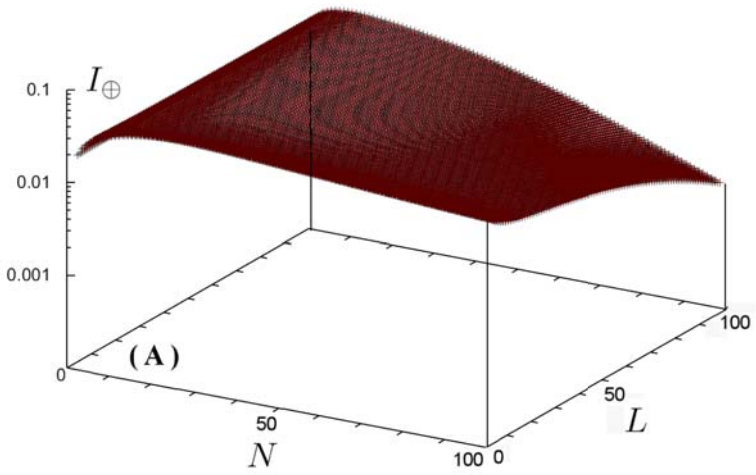

Figure 2.2: Active Information per query for single-mutation from section 2.2.3 The axis labeled L is the size of the alphabet. The axis labeled N is the length of the message. The axis labeled $I_\oplus$ is the average active information per query.

(3) **Staying at state λ.** With (2.12), all transitions from the λ node in the center entry in Figure 2.1 are now identified.

2.2.3.3 *Results* For any particular choice of message length, L, and alphabet size, N, there will be a choice of the number of offspring, K, which gives the smallest number of generations to find the hidden string. Figure 2.2 shows a plot of the active information per query where the choice of $K = 100$.

2.2.4 Ratchet Strategy.

This strategy is similar to the previous strategy except that only one child is generated per generation. A single mutation at a randomly selected location is performed. If the child has a better Hamming distance than the parent, it is kept. Otherwise the parent is asked to generate another child and the process is repeated.

2.2.4.1 *One Child Ratchet Analysis* The method of mutation is the similar to the one child mutation in Section 2.2.3(A). As before, we evaluate three cases.

(1) **Worse.** It is impossible to do worse, since any deleterious mutation will be rejected. Thus $\Pr[W] = 0$.

(2) **Better.** The probability of a better score is the same as before in (2.8)

$$\Pr[B] = \frac{L - \lambda}{NL}. \tag{2.13}$$

(3) **Same.** The probability of the same score can be derived by adding (2.9) and (2.7), since the score will remain the same if the mutation is either deleterious or neutral.

$$\begin{aligned} \Pr[S] &= \frac{(N-1)\lambda}{NL} + \frac{\lambda + (N-1)(L-\lambda)}{NL} \\ &= \frac{\lambda + (N-1)L}{NL}. \end{aligned} \tag{2.14}$$

2.2.4.2 *Markov Matrix.* As before we can use the probabilities to construct a Markov matrix. The chain is similar to that in the previous case except that there is no way to move backwards.

(1) **Coming from state $\lambda - 1$.** With $\lambda \to \lambda - 1$, (2.13) gives

$$\Pr[\lambda|\lambda - 1] = \frac{L - (\lambda - 1)}{NL}$$

(2) **Staying at state λ.** From (2.14),

$$\Pr[\lambda|\lambda] = \frac{\lambda + (N-1)L}{NL}$$

The exceptions are (1) $\lambda = 0$ for which $\Pr[\lambda|\lambda - 1] = 0$, and (2) the absorbing state $\lambda = L$ for which $\Pr[\lambda|\lambda] = 1$.

2.2.4.3 *Results* Figure 2.3 shows the active information per query for the ratchet strategy given different alphabet sizes and message lengths. The AIPQ, and

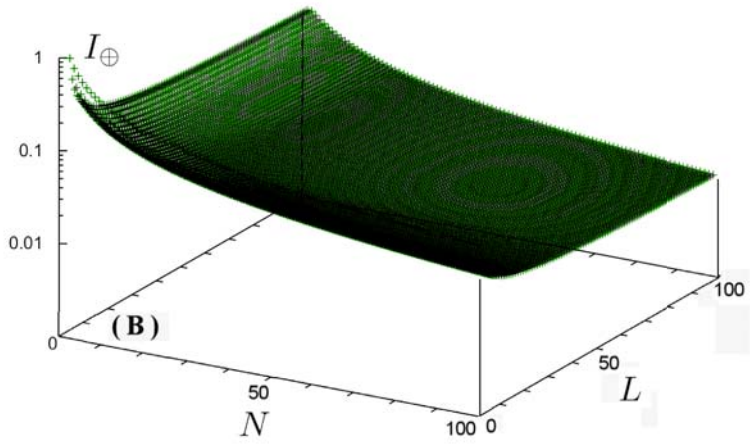

Figure 2.3: Ratchet evolutionary strategy from Section 2.2.4 The axes are the same as in Figure 2.2

thus efficiency of information extraction, declines as both the message length and alphabet size are increased. The plot, however, shows better results than those in Figure 2.2.

2.2.5 Mutating Children With a Fixed Mutation Rate.

In this strategy each letter in the string has a probability of being changed. Each child will have the same string as the parent except that each letter will be changed with a fixed probability, μ.

2.2.5.1 *For One Child* We use the same notation as in Section 2.2.3(A) and add M for *mutated*. Thus $\bar{M}$ means *not mutated*.

(1) Define the probability of *worse from good* for a single character as

$$\begin{aligned} p_w &:= \Pr[W|g] \\ &= \Pr[(W|g)|M]\Pr[M] + \Pr[(W|g)|\bar{M}]\Pr[\bar{M}] \\ &= \frac{N \quad 1}{N}\mu + 0 = \frac{\mu(N \quad 1)}{N} \end{aligned} \tag{2.15}$$

Then the *same from good* probability is $1 - p_w$. We have λ good characters. Let k_w denote the number of the λ good characters that become bad. Then, for $0 \leq \omega \leq \lambda$,

$$\begin{aligned} \Pr[k_w = \omega] &= \binom{\lambda}{\omega} p_w^{\omega} (1 - p_w)^{\lambda - \omega} \\ &= \binom{\lambda}{\omega} \left(\frac{\mu(N-1)}{N} \right)^{\omega} \\ &\quad \times \left(1 - \frac{\mu(N-1)}{N} \right)^{\lambda - \omega} \end{aligned} \tag{2.16}$$

(2) Define the probability of *better from bad* as

$$\begin{aligned} p_b &:= \Pr[B|\bar{g}] \\ &= \Pr[(B|\bar{g}|M]\Pr[M] + \Pr[B|\bar{g}|\bar{M}]\Pr[\bar{M}] \\ &= \frac{1}{N}\mu + 0 = \frac{\mu}{N}. \end{aligned} \tag{2.17}$$

Then the probability of *same from bad* is $1 - p_b$.

We have $L - \lambda$ bad characters. Let k_b denote the random variable of the number of $L - \lambda$ characters that become good. Then, for $0 \leq \beta \leq L - \lambda$,

$$\begin{aligned} \Pr[k_b = \beta] &= \binom{L-\lambda}{\beta} p_b^{\beta} (1 - p_b)^{(L-\lambda)-\beta} \\ &= \binom{L-\lambda}{\beta} \left(\frac{\mu}{N} \right)^{\beta} \left(1 - \frac{\mu}{N} \right)^{(L-\lambda)-\beta}. \end{aligned}$$

The change in Hamming distance is $\Delta\lambda = k_b - k_w$. The probability that $\Delta\lambda = \kappa$ is

$$f_{\Delta\lambda}(\kappa) := \Pr[\Delta\lambda = \kappa] = \sum_{\beta - \omega = \kappa} \Pr[k_b = \beta]\Pr[k_w = \omega]$$

which, for $-\lambda \leq \kappa \leq L - \lambda$, can be written

$$\begin{aligned} f_{\Delta\lambda}(\kappa) &= \sum_{\beta=\max(0,\kappa)}^{\min(\kappa+\lambda, L-\lambda)} \binom{L-\lambda}{\beta} (\mu\pi)^{\beta} \\ &\quad \times (1-\mu\pi)^{(L-\lambda)-\beta} \\ &\quad \times \binom{\lambda}{\beta-\kappa} (\mu(1-\pi))^{\beta-\kappa} \\ &\quad \times (1-\mu(1-\pi))^{\lambda-(\beta-\kappa)} \end{aligned} \tag{2.18}$$

where $\pi = 1/N$ see Figure 2.4

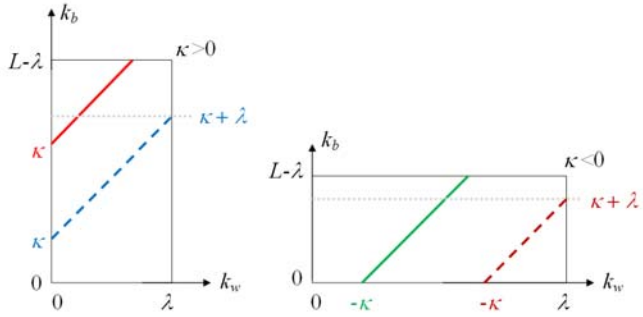

Figure 2.4: Determining the limits on the summation
See equation 2.18. On the left is $\kappa > 0$ and the right is $\kappa < 0$. The lines shown are $k_b - k_w = \kappa$.

From (2.18) the cumulative distribution of $\Delta\lambda$, is

$$\begin{aligned} F_{\Delta\lambda}(x) &= \Pr[\Delta\lambda \leq x] \\ &= \begin{cases} 0 & ; \quad x < -\lambda \\ \sum_{\kappa=-\lambda}^{x} f_{\Delta\lambda}(\kappa) & ; \quad -\lambda \leq \kappa \leq L-\lambda \\ 1 & ; \quad x > L-\lambda. \end{cases} \end{aligned} \tag{2.19}$$

2.2.5.2 *For K Children.* For K kids, let the change in the kth child be $\Delta\lambda_k$. The

largest change is

$$\Delta\lambda_{\max} = \max_{k=1}^{K} \Delta\lambda_k$$

and

$$F_{\Delta\lambda_{\max}}(x) = \Pr[\Delta\lambda_{\max} \leq x] = [F_{\Delta\lambda}(x)]^K \tag{2.20}$$

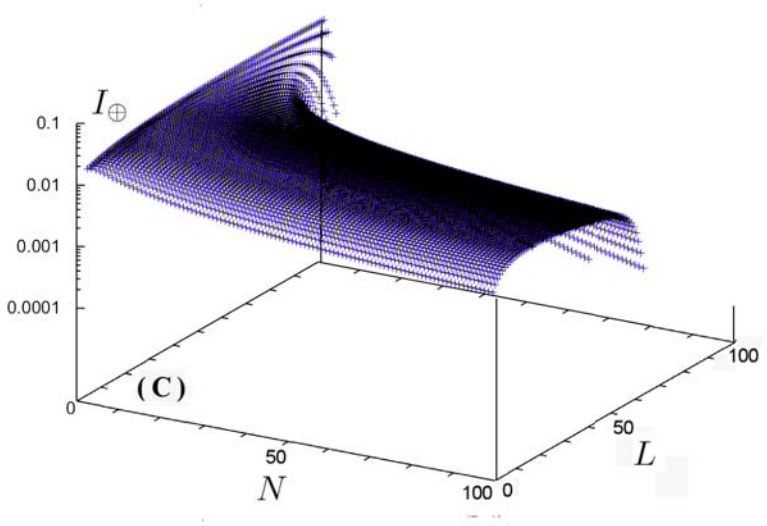

Figure 2.5: Fixed mutation rate ($\mu = .05$) from section 2.2.5
The axes are the same as in Figure 2.2

The corresponding probability mass function is

$$f_{\Delta\lambda_{\max}}(x) = F_{\Delta\lambda_{\max}}(x) \quad F_{\Delta\lambda_{\max}}(x \quad 1). \tag{2.21}$$

2.2.5.3 *Markov Matrix* The probability mass function in (2.21) can be explicitly parameterized as $f_{\Delta\lambda_{\max}}(x, \lambda)$. For the mutation example, the Markov matrix then has elements

$$p_\lambda, l, k = f_{\Delta\lambda_{\max}}(\lambda, k).$$

2.2.5.4 *Results* As illustrated in Figure 2.5, this algorithm has a slow decline in information per query until it suddenly collapses when the mutation rate becomes too high. If, for example, we are within a single character of identifying a phrase, then a mutation rate that gives on average, say, 10 mutations will take a long time to take the final small step to perfection.

2.2.6 Optimizing the Mutating Schedule for K Children.

Rather than always using a constant mutation rate, we can select the optimal mutation rate for each generation. We can optimize the mutation rate by maximizing the expected value of $\Delta\lambda_{\max}$ as a function of μ. Using (2.20) and (2.21)

$$\begin{aligned} E\left[\Delta\lambda_{\max}\right] &= \sum_{x=-\lambda}^{L-\lambda+1} x f_{\Delta\lambda_{\max}}(x) \\ &= \sum_{x=-\lambda}^{L-\lambda+1} x \left[F_{\Delta\lambda_{\max}}(x) - F_{\Delta\lambda_{\max}}(x-1)\right] \\ &= \sum_{x=-\lambda}^{L-\lambda+1} x \left[F^K_{\Delta\lambda}(x) - F^K_{\Delta\lambda}(x-1)\right] \end{aligned} \tag{2.22}$$

where $F_{\Delta\lambda}(x)$ is defined by (2.18) and (2.19).

2.2.6.1 *Results* For each value of L and N, we have optimized [4] the choice of μ to maximized (2.22). The active information per query as a function of mutation rate, μ for each value of λ. The results are shown in Figure 2.6(D).

2.3 Frequency of Occurrence Hamming Oracle Algorithm

The Markov models use only the current state to determine the next step in the search. There is no attempt made to use the history of the search. The *frequency of occurrence (FOO) Hamming oracle algorithm* (FOOHOA) does. As one might expect, the more knowledge a search procedure can effectively use, the greater the resulting active information per query.

The FOOHOA is an efficient algorithm to find the hidden string in a Hamming oracle which, unlike the previous stochastic hill-climbing search methods, establishes and updates a FOO (Dembski and Marks, 2009b) knowledge base. We begin by establishing the FOO of the string. If a string containing all A's is submitted to a

[4] An attempt at maximizing (2.22) by differentiation with respect to μ and setting to zero results in analytic difficulty. Numerical evaluation of (2.22) is more straightforward.

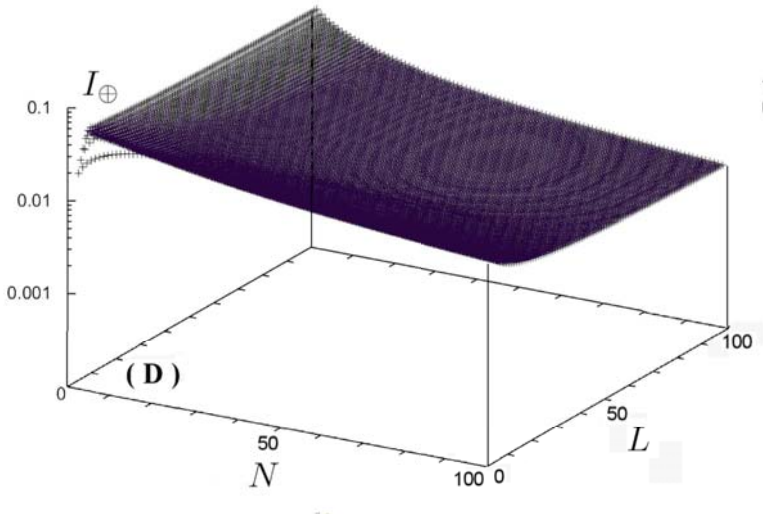

Figure 2.6: Active information per query, $I_\oplus$ (in bits), for an evolutionary strategy using optimally scheduled mutation rates presented in section 2.2.6
The axes are the same as in Figure 2.2

Hamming oracle, the oracle's response will allow the algorithm to determine how many A's are in the hidden string. By repeating this process with all of the letters in the chosen alphabet, we determine the FOO for all of the letters. If there are N characters in the alphabet, establishment of the FOO requires, at most, $N \quad 1$ queries.[5] In rare instances, of course, the target might be identified with the intially established FOO.

The remainder of the FOOHOA is best explained by example. Consider a Hamming oracle using the English letters as its alphabet and having a message length of 5. Under this algorithm, we will already know the oracle's response to `AAAAA`, because we have already establish the FOO for all letters. Consider the query `ABAAA`.

- If the second letter in the hidden string is `A`, the distance will increase.

[5] An even more efficient procedure is to present letters in their frequency of occurrence in the English language. The space would be first presented followed by the most commonly used English letter, E. For shorter phrases, infrequently used letters, like Q and Z, will not occur with high probability. Knowing this letter ordering is additional knowledge that will on average increase the per query active information. For our implementation, we do not use this knowledge and, instead, simply proceed through the alphabet from A to Z.

- If the second letter in the hidden string is `B`, the distance will decrease.
- Otherwise, the distance will remain the same.

The query in question will actually test the second position both for presence of A and B. The FOO Hamming oracle algorithm uses this principle for all queries after establishing the frequency of occurrence. It starts on the left side of the string and works through the string, querying each letter in order from the FOO list until it discovers the correct letter. Letters are tested starting with the most frequent because they have the largest probability of being in any unfilled position. Once the correct value of a letter has been established, the FOO table is updated by omitting the contribution of the identified character.

Algorithm 1 FOOHOA Algorithm

```
for all c where c is a letter in the alphabet do
  FOO[c] ← query( c repeated l times)
  CURRENT_FOO[c] ← FOO[c]
end for
for i = 1 to l do
  while correct letter for position i has not been found do
    if only one letter with CURRENT_FOO > 0 exists then
      message[i] ← letter
    else
      x ← untried letter with highest CURRENT_FOO
      y ← untried letter with highest CURRENT_FOO
      q ← a string with l x's, but with position i having an y
      if query(q) < FOO[x] then
        message[i] ← x
      else if query(q) > FOO[x] then
        message[i] ← y
      end if
    end if
  end while
end for
return message
```

The average active information per query, $I_{\oplus}$, for the FOO Hamming oracle algorithm is shown in Figure 2.7 as a function of alphabet size, N, and message

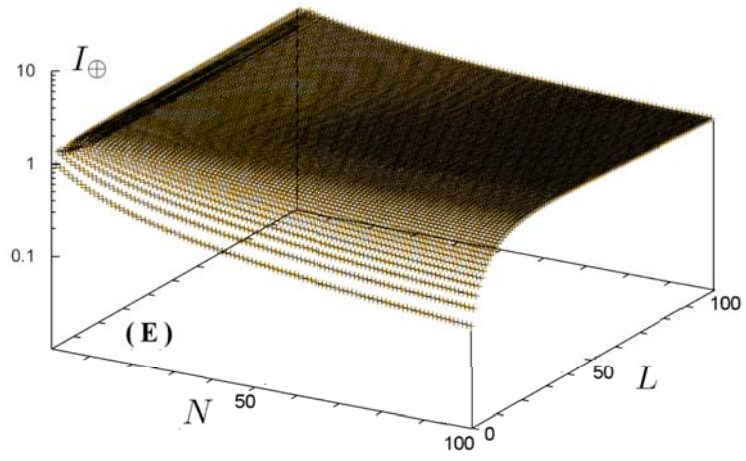

Figure 2.7: Performance of the Frequency Hamming Oracle Algorithm presented in section 2.3
The axes are the same as in Figure 2.2

length, L. The results are significantly better than the Markov results shown in the other plots in Figures 2.2, 2.3, 2.5, and 2.6.

Figure 2.8 shows the active information per query average across different message lengths for various alphabet sizes. The FOOHOA is the most effective algorithm for information extraction measured by queries.

2.4 Optimal Hamming Search: A Search for the Search

For a given oracle, there exists an algorithm that, on average, extracts the maximum active information per query. For the Hamming oracle, we are able to search for the optimal algorithm. In general, a *search for a search* (S4S) is exponentially more computationally demanding than a search itself (Dembski and Marks, 2009a, 2010b). Using an exhaustive inspection of all possible search trees, we generate an optimal tree search in the sense of maximum extraction of per query active information from the oracle. The maximum average active information is obtained when the search tree has minimal average depth.

$\downarrow L\ N \rightarrow$	1	2	3	4	5	6
1	0	1.000	1.667	2.250	2.800	3.333
2	0	1.500	2.337	3.125	3.281	4.611
3	0	2.250	2.889	3.822	-	-
4	0	2.750	3.469	-	-	-
5	0	3.375	-	-	-	-
6	0	3.875	-	-	-	-

Table 2.1: Expected number of queries, Q, from the search algorithm found in the S4S over all search trees.

$\downarrow L\ N \rightarrow$	2	3	4	5	6
1	1.000	0.951	0.889	0.829	0.775
2	1.333	1.359	1.280	1.415	1.121
3	1.333	1.646	1.570	-	-
4	1.454	1.827	-	-	-
5	1.481	-	-	-	-
6	1.548	-	-	-	-

Table 2.2: Active information per query, $I_{\oplus}$, from the search algorithm found in the S4S over all search trees.

At each iteration of a search algorithm, there is some set of possible hidden strings which have not been ruled out by previous queries. The algorithm selects a query as some function of this set. The resulting query and the response will produce a new subset containing only the strings that are compatible with the new query result. We want to find the function mapping these sets to queries that will result in the lowest average number of queries to determine the target. We can do so by searching every possible function to find the optimal one. This is an exhaustive S4S performed on the original search space. It should not be surprising, therefore, that the search is very expensive and can only be run for very small problems.

Tables I and II shows the best possible active information per query across different message lengths for various alphabet sizes. On a per query basis, information cannot be extracted more efficiently.

2.5 Conclusions

By exploring a variety of algorithms we have demonstrated that the simple Hamming oracle can be used with surprisingly different degrees of efficiency as measured by query count. The FOO Hamming oracle algorithm manages to extract approximately one bit of information per query indicating that the oracle can be a significant source of information. In comparison, evolutionary search as modeled by Markov processes uses the Hamming oracle inefficiently. The success of a searches derives not from any intrinsic property of the search algorithm, but from the information available from the oracle as well as the efficiency of the search algorithm in the extraction of that information.

A high level interactive simulation of algorithms showing their varying effectiveness in extracting active information using a Hamming oracle is available on line at `http://www.EvoInfo.org/`.

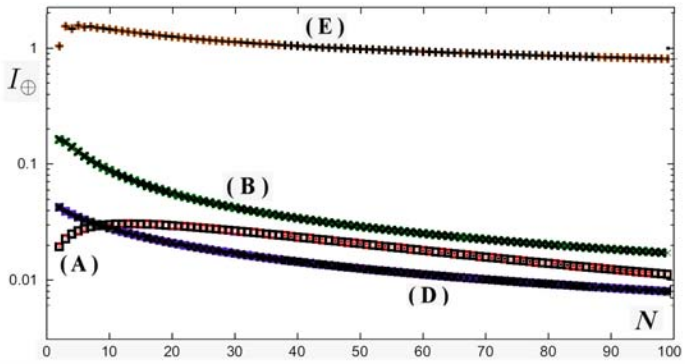

Figure 2.8: Averaged active information per query as a function of alphabet size, N Averaged active information per query, $I_\oplus$ (in bits), for various algorithms as a function of alphabet size, N. (A) single-mutation presented in section 2.2.3(A); (B) ratcheted single mutation in section 2.2.3(B) for one child. (D) Active information per query, $I_\oplus$, for an evolutionary strategy using optimally scheduled mutation rates presented in section 2.2.3(D) for 100 children; (E) Performance of the FOO Hamming oracle algorithm presented in section 2.3. These plots are obtained by averaging over $1 \leq L \leq 100$. The data from section 2.2.5 is not included because the problem analysis becomes ill-conditioned in places and not all data is available.

CHAPTER THREE

Ev [1]

In this section, we analyze a search algorithm dubbed Ev that models the evolution of nucleotide binding sites (Schneider, 2000). The Ev search can be viewed (Strachan, 2003) as an inversion (Jensen et al., 1999) of a perceptron (Reed and Marks II, 1999). This is illustrated in Figure 3.1. There are $n = 256$ nucleotides at the perceptron output that serve as potential binding sites [2] each with $\Gamma = 4$ possible bases (A, G, C and T). Under an assumption of uniformity, each base therefore corresponds to 2 bits of information. In this sequence, 16 of the 256 nucleotide sites are designated binding sites. The Ev simulation uses randomly assigned binding sites, which are then fixed for the duration of a run. A published example (Schneider, 2000) uses the specific locations[3]

$$t = [1, 10, 17, 26, 33, 43, 50, 60, 70, 76, 83, 92, 101, 109, 117, 125]. \tag{3.1}$$

There is also a $\Gamma \times \lambda$ matrix of weights with $\Gamma\lambda = 24$ elements that are restricted to integers in the range $[-R, R-1]$ with $R = 512$. There is also a single bias, θ, which has the same range as the weights. Each of the binding-site locations is assigned a

[1] A previous version of this chapter was published as: Montañez G, Ewert W, Dembski WA, Marks II RJ (2010) A vivisection of the ev computer organism: Identifying sources of active information. BIO-Complexity 2010(3):1-6. doi:10.5048/BIO-C.2010.3. Much of this chapter, including figures, are taken verbatim from this work The original paper is published on an online journal under the Creative Commons Attribution License. See http://bio-complexity.org/.

[2] Although all 256 positions along the genome are evaluated for errors, the randomly placed binding sites are restricted to the second half of the genome. In the first Figure of (Schneider, 2000), these correspond to bases 126 to 261. There are other nucleotides whose identity is interpreted as weights, window values, or the bias in the construction of the perceptron. Five additional bases are used at the end to accommodate a sliding window used in Ev.

[3] These binding sites in start at location 131 (zero-indexed) in the first figure of (Schneider, 2000). Thus, location 10 here corresponds to nucleotide 141 in the first figure of (Schneider, 2000).

value of one and all other sites a value of zero. The search problem is to find the target binding sites, such as those in (3.1).

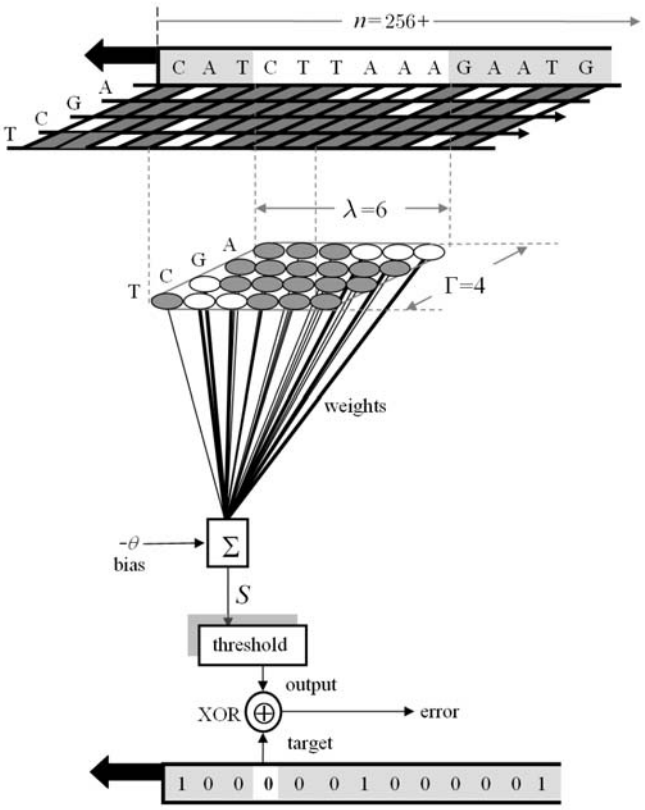

Figure 3.1: The search structure used by Ev is numerically equivalent to inversion of a perceptron.

Our purpose is not to discuss the biology motivating Ev, but to analyze how its search structure constrains performance in light of conservation of information theorems for search algorithms. There are at least two interpretations of the search.

3.1 Direct Search for the Target from the Viewpoint of the Output

The accumulated error from the XOR in Figure 3.1 is a *Hamming oracle* for a binary alphabet which announces the total number of bits the target differs from the output. The Hamming oracle is a rich source of active information (Chapter 2).

Searching only from the perspective of the output, the binary target can be identified with an AIPQ of $I_{\oplus}$ = one bit or more active information per query.[4]

3.2 Target Search Through the Perceptron Structure

Our analysis deals with the more interesting case where the perceptron output is not independently controlled, but is rather the response to a stimulus of Ev's perceptron structure. For a given weight matrix, bias, genome sequence of bases, and bit stream target, Ev can be interpreted using the time series perception (Reed and Marks II, 1999) in Figure 3.1. The genome on the top has at each position $\Gamma = 4$ binary locations, each corresponding to A, G, C and T. The number one is inserted at the location corresponding to the base type, and the number zero is inserted at each of the three remaining locations. These are illustrated at the top of Figure 3.1 with white squares for 1's and shaded squares for 0's. These present binary (0,1)-inputs to the perceptron denoted in Figure 3.1 where circles are shaded like the squares above them. The strip of the genome is marched from right to left in unit increments of time. At each point in time, the window of $\lambda = 6$ bases with binary inputs is multiplied by the weights. The values are added. Then the bias θ is subtracted. If the final sum is positive, the output is a one and otherwise it is a zero. The output is compared to the target. If they are different, an error is tallied. The genome sequence and the target sequence of (0,1)'s is advanced one step, and the process is repeated for the next target bit.

In the search for the binding sites, the target of zeros and ones is fixed. The weights, genome sequence, and bias are all allowed to vary. Finding them is a form of the perceptron inversion problem (Jensen et al., 1999). A search space Ω, consists of all possible values of weights, genome sequence, and bias. There are 256 nucleotide

[4] A simple approach is this. Query with an output of all ones and record the Hamming distance. Change the first bit to zero. If the Hamming distance is larger, the first bit is a one. If smaller, a zero. Repeat for all the bits. When the second last bit is identified, the final bit can be identified by matching the Hamming distance measured first. This approach supplies one bit of active information per query. We can do even better.

bases (A,C,G,T) used for each simulation of Ev. There are 5 additional bases used for a boundary condition for a sliding window across some of the bases. Including these bases, the search space is the 261 fold Cartesian product of the (A,C,G,T) bases. Bases are interpreted in different ways in the search.

(1) As a nucleotide on the input strip at the top of Figure 3.1. There are $256 + \lambda - 1 = 261$ nucleotides in the input strip.

(2) As a weight in the $\Gamma\lambda$ matrix. Each weight has 10 bits (5 bases) of accuracy.

(3) The bias threshold, θ, can be altered. The bias also has 10 bits of accuracy.

Since there are four bases, the cardinality of the search space[5] is thus

$$|\Omega| = 4^{261} = 1.37 \times 10^{157}. \tag{3.2}$$

Every point in Ω generates an output of 256 bits. The target T, is any point in Ω that generates the 256 bit nucleotide binding site sequence at the output. Initialization is always random.

One's first inclination is to announce an endogenous information of $I_\Omega = 256$ bits. This, however, is wrong if we choose the search space Ω. Each element in Ω, rather, is assigned a vector of length $L = 256$ bits. Some of the assigned vectors are equal to the target binding site vector. Most are not. The fraction of elements of Ω assigned the correct target is equal to the probability, p, of success of an unassisted search. We bound the endogenous information in Section 3.3.2 by $I_\Omega < 90$ bits.

Having established the perceptron structure of Ev, we now investigate algorithms, $\mathcal{A}$, to perform successful search.

3.2.1 Stochastic Hill Climbing - Algorithm $\mathcal{A}_1$

Stochastic hill climbing can be performed at the perceptron level involving all 261 of the genome bases in Ev. We choose, at random, 261 genome bases which generates the output in Figure 3.1 from which the error is evaluated. One of the

[5] Including the five bases that are not searched.

261 bases is replaced at random and the error calculation is repeated. If the error is the same or smaller, we keep the change. If not, the change is discarded and the process repeated.[6] Simulation of $K = 10,000$ separate searches[7] with a query limit, $\acute{Q} = 100{,}000$ and random initializations produced a 100% success rate. The average number of queries for a success was $E[Q] = 10,601$. We obtained the normalized active information (NAIPQ) by divising the active information by the endogenous information. The NAIPQ for algorithm $\mathcal{A}_1$ is

$$\frac{\mathcal{I}_{\oplus_1}}{I_\Omega} = 1.09 \times 10^{-4} \tag{3.3}$$

3.2.2 Evolutionary Perceptron Inversion of Ev- Algorithm $\mathcal{A}_2$

Ev originally used an evolutionary search algorithm for finding the binding sites (Schneider, 2000). The search is seeded with $M = 64$ randomly selected *organisms* and, in each iteration, discards two adjacent bits from a randomly chosen nucleotide in each organism, replacing them with two randomly chosen bits. Half of the mutated genomes with the highest fitness are selected to be the parents of the next generation. (*i.e.*, (32,64)-ES (Phanendra et al., 1994; Bäck and Schwefel, 1993).)

Using $\acute{Q} = 100{,}000$ as the maximum query count per search,[8] simulation of $K = 10{,}000$ separate searches using randomly generated initializations produced 9,115 successes. When there was a success, the average number of queries was $\langle Q \rangle = 63,568$, or 993 *generations*. This compares to the single simulation result of 704 generations reported in the original Ev paper (Schneider, 2000). The NAIPQ using **??** for algorithm $\mathcal{A}_2$ is

$$\frac{\mathcal{I}_{\oplus_2}}{I_\Omega} = 1.55 \times 10^{-7}$$

[6] This algorithm is commonly denoted as (1+1)-ES (Phanendra et al., 1994; Bäck and Schwefel, 1993).

[7] All experimental results in this paper were obtained using the online Ev simulation software available at `http://www.evoinfo.org/ev`.

[8] 10,000 populations each running for a maximum of 1,563 generations (corresponding to $\acute{Q} = 64 \times 1,563 \approx 100,000$ queries)

Comparing with (3.3), we see that algorithm $\mathcal{A}_1$ is about 7 times more efficient in terms of the number of queries required.

3.2.3 Differing Mutation Rates

The above results were obtained using a fixed mutation rate of one base change (two bits) per organism per generation for both $\mathcal{A}_1$ and $\mathcal{A}_2$. We further measured search performance of the $\mathcal{A}_2$ strategy using several different mutation rates.[9] A comparison of success rate and mutation rate is shown in Figure 3.2 for the $\mathcal{A}_2$ strategy. The optimal mutation rate for the $\mathcal{A}_2$ strategy was found to be roughly 1.75 mutations per child,[10] per generation, with the 1 mutation per child rate chosen by Schneider giving fairly good results.

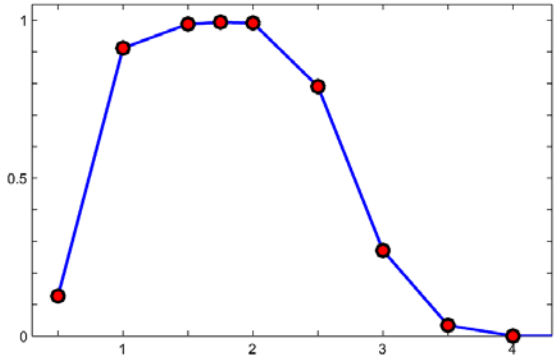

Figure 3.2: Plot of search success rate versus mutation rate for algorithm $\mathcal{A}_2$ with population size of 64 and query cutoff of $\acute{Q} = 100{,}000$ queries.

3.2.4 Random Number of Binding Sites

We show in Section 3.3 that the perceptron favors generation of strings of output zeros peppered with occasional ones, or ones peppered with zeros. The binding site targets tend to be the former. Experiments demonstrated a severe

[9] All tests were performed for 10,000 runs, using a population size of 64 and a query cutoff of 100,000 queries.

[10] Fractional mutations are generated by randomizing the mutation number. To achieve 1.75 mutations per child, each child receives at least one mutation and has a 75% chance of receiving an additional, second mutation.

performance decrease when the binding sites were chosen by a 50-50 chance method. We simulated the Ev search using randomly generated binding sites, with each site along the second half of the genome[11] having a 50% chance of being marked as a binding site. We measured the performance using mutation rates of 1, 1.5, 2, 4 and 8 mutations per child, per generation with a query cutoff of 100,000 queries, for 10,000 runs each. The Ev search was only successful for mutation rates of 1 and 1.5, failing to find the target in under 100,000 queries for all other mutation rates. A mutation rate of 1 mutation per child using the random 50-50 bindings resulted in a success rate of only 0.05 (compared to the prior success rate of 0.91; an 86% decrease in performance), while a mutation rate of 1.5 mutations per child resulted in a success rate of 0.0003 (compared to the prior success rate of 0.99.) Changing the binding sites pattern to an even mix of zeros and ones severely hampered search performance. The reasons for this will be discussed in the next section.

3.3 Performance Analysis of the Ev Perceptron

A source of the active information in Ev is the structuring of the perceptron in Figure 3.1. With reference to this figure, denote the sum entered into the threshold operator in Figure 3.1 by S. There are 256 values of S. Let's call the nth sum S_n and concatenate them into the vector

$$\vec{S} = [S_1\ S_2\ S_3\ ...\ S_n\ ...\ S_{256}\ ]^T \tag{3.4}$$

A given S_n is the sum of seven i.i.d. uniform discrete random variables. The random variables in $\vec{S}$, though, are far from independent. Each, for example, contains the same bias, θ, as one of the seven numbers in its sum. If the output bits were independent, every possible binary string would have the same probability as any other and the same distribution of error. According to the Laplace-DeMoivre theorem,

[11] Positions 126 through 256

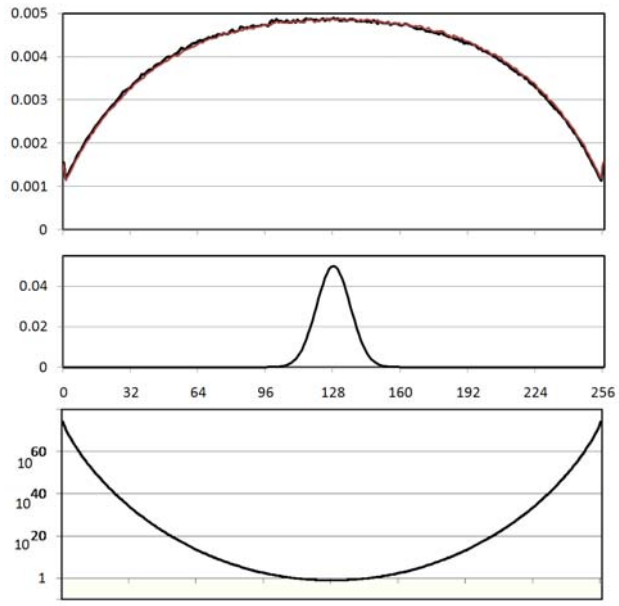

Figure 3.3: Plots showing the bias of the inverted perceptron
The range of all three plots is from zero to 256. TOP: The normalized histogram of the errors in Ev for ten million trials for an all ones target (lighter line) and ten million trials for all zeros (darker line) are nearly graphically indistinguishable. For both there are 0.15% of the cases where there are zero errors. MIDDLE: The Gaussian distribution we expect from the Laplace-DeMoivre theorem. BOTTOM: This curve, equal to the top curve divided by the middle curve, tells the degree to which the relative frequency has been amplified by Ev's perceptron. The frequency of all zeros expected by a randomly chosen string of bits is multiplied by 1.8×10^{74} by the perceptron. The frequency of occurrences of exactly 16 ones is amplified by a factor of 2.8×10^{49}.

they would therefore be Gaussian as in the middle plot of Figure 3.3. The normalized histogram at the top of Figure 3.3 is far from Gaussian.

If θ is large and positive, it can push all of the elements in $\vec{S}$ to be positive. Once thresholded, all of the positive values become one. Likewise, the opposite polarity of the bias will push the outputs of the threshold to zero. Only with biases near zero will the resulting binary string be a nearly even mix of ones and zeros. Outcomes with a large number of ones and a few zeros, or few ones and many zeros, are therefore more probable outcomes of the perceptron.

3.3.1 No Site a Binding Site

Analyzing the output consisting of all zeros (*i.e.*, no site is a binding site) illustrates the tendency for the perceptron to produce certain outputs more frequently than others.

To see this, run the perceptron in Figure 3.1 once using random sampling with no iteration, and assess the Hamming distance between the perceptron output bits and the target of all zeros. The process is repeated using freshly generated random numbers. Figure 3.3 shows a histogram of the Hamming distance between randomly generated perceptron outputs and the target for ten million such trials when the output is all zeros. Rather than dipping to a near zero at the origin, the empirical probability of success for the error of getting a zero (instead of a one) is

$$q = 0.00155. \tag{3.5}$$

The endogenous information for an output of all zeros is therefore $I_\Omega = -\log_2(q) = 9$ bits. Similar experiments can be performed for an all ones target, with near identical results.[12] The decrease in endogenous information is due to active information introduced by the perceptron structure, for an all zeros target. The perceptron therefore adds active information to the general Ev search. The reason, as we show, is that the Ev perceptron is heavily predisposed for generating successful solutions for targets of the type in (3.1).

There are a total of 261 nucleotides that define the perceptron structure in Figure 3.1 (Schneider, 2000) and the size of the search space is given by (3.2). For the case of all zeros at the output, an astounding $q|\Omega| = 2.12 \times 10^{154}$ produce an output of all zeros. This results takes into account the combination of the bias, weights, and genome sequence.

[12] 15,234 of 10 million trials. For all ones, $q = 0.00152$

The number of repeated Bernoulli trials with replacement until a success occurs for all ones is a geometric random variable, Q (for queries), with mean

$$\mathbb{E}[Q] = \frac{1}{q} = 645 \text{ random queries.} \tag{3.6}$$

This is the expected number of queries in the all zeros case for a perfect search.

3.3.2 An Upper Bound on Endogenous Information

Despite this predisposition, the probability of an unassisted search generating zeros with sparsely occurring ones, as specified for example in (3.1), is still very small. We have two empirical estimates of the upper bound on the endogenous information of the Ev problem. In both cases we conservatively assume there is but one successful perceptron that generates a desired 16 bit target *e.g.* as in (3.1).

(1) From the data used to plot the top figure in Figure 3.3, the frequency of occurrence of strings with zeros with 16 ones is

$$\Pr[\text{16 ones in 256 bits}] = 0.0024.$$

The probability of hitting the target in (3.1) is then

$$p = \Pr[\text{ hitting target }] = \Pr[\text{ hitting target } |\text{16 ones}]\Pr[\text{16 ones}].$$

There are

$$\frac{1}{\pi_1} = \binom{256}{16} = 10^{25}$$

possible ways to arrange 16 ones and 240 zeros. Because there have been successful simulations, we know at least one of these can be the target. Thus $\Pr[\text{ hitting target } |\text{16 ones}] \geq \pi_1$ and $p \geq 0.0024 \times \pi_1 = 2 \times 10^{-28}$. Thus

$$I_\Omega < 92 \text{ bits.} \tag{3.7}$$

(2) Additionally, Ev requires that all binding sites be restricted to the second half of the genome[13] as they are in the target sequences. After randomly

[13] Positions 126 through 256

initializing 1.5 billion independent genomes, 187 outputs were found to have exactly 16 ones occurring along the second half of the genome. Repeating the above calculations[14] resulted in the slightly tighter bound

$$I_\Omega < 90 \text{ bits.} \tag{3.8}$$

3.3.3 Sources of Information in Ev

There are at least five sources of active information in Ev.

a) *The perceptron structure.* The perceptron structure is predisposed to generating strings of ones sprinkled by zeros or strings of zeros sprinkled by ones. Since the binding site target is mostly zeros with a few ones, there is a greater predisposition to generate the target that if it were, for example, a set of ones and zeros produced by the flipping of a fair coin. See Section 3.3 for details.

b) *The Hamming Oracle* . When some offspring are correctly announced as more fit than others (Mackay, 2004), external knowledge is being applied to the search and active information is introduced. As with the child's game, we are being told with respect to the solution whether we are getting "colder" or "warmer".

c) *Repeated Queries.* Two queries contain more information than one. Repeated queries can contribute active information (Dembski and Marks, 2009b).

d) *Optimization by Mutation.* This process discards mutations with low fitness and propagates those with high fitness. When the mutation rate is small, this process is a simple Markov birth process (Papoulis, 1991) that converges to the target (Dembski and Marks, 2009b).

e) *Degree of Mutation.* As seen in Figure 3.2, the degree of mutation for Ev must be tuned to a band of workable values.

[14] Using $\binom{131}{16}$ to reflect our smaller number of potential binding site positions.

3.4 Conclusion

The success of Ev is largely due to active information introduced by the Hamming oracle and from the perceptron structure. As discussed in Chapter 2, the Hamming Oracle is an effective source of knowledge that a search algorithm can extract information from. In addition, the bias towards the target introduced by the perceptron structure constitutes information which helps the search locate the target. The search algorithm used by Ev is not even the most effective strategy given its available knowledge sources as the ratchet strategy easily outperformed it.

CHAPTER FOUR

Avida: Evolutionary Search Using Nand Logic [1]

The information measures of many search algorithms are beyond direct analytic evaluation and require empirical analysis using, for example, Monte Carlo simulation. One is evolutionary search for synthesis of logic functions. Iand gate is one of two logic functions from which all other logic can be synthesized (Hein, 2010). The other is the nor gate. Chips containing a *sea of gates* (de Lima and Kinniment, 1995; El Gamal et al., 1989; Hanibuchi et al., 1991), all nand, are therefore capable of universal logic. This property allows evolutionary development of logic functions using the nand gate as the single logic component (Chan, 1994; Golic, 2007; Takita and Kakazu, 1998, 1999; Yasunaga et al., 1999).

Avida (Lenski et al., 2003), illustrated in Figure 4.1, performs logic synthesis using only the nand gate. The motivation behind Avida is not engineering design, but is rather to (Lenski et al., 2003)

> "... show how complex [biological] functions can originate by random mutation and natural selection."

Avida generates an output equal to a logic combination of the inputs, X and Y. The logic operation under consideration that requires the most nand gates (five) is the XNOR. This is the ultimate goal of the search. The example of Avida in Figure 4.1 has performed the XNOR. The first bits X and Y are 1 and 0 and the XNOR of 1 and 0 is 0. This then is the first bit of the output string. The second bits of X and Y are both ones and the XNOR of two ones is 1. This is the second bit

[1] A previous version of this chapter was published as: Winston Ewert, William A. Dembski and Robert J. Marks II, "Evolutionary Synthesis of Nand Logic: Dissecting a Digital Organism," Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics. San Antonio, TX, USA - October 2009, pp. 3047-3053. The copyright notice is in Appendix A. Much of this chapter, including figures, is taken verbatim from this work.

in the output. Continuing with subsequent bits, we say the machine in Figure 4.1 generates an XNOR if the bitwise XNOR's of X and Y is equal to the corresponding output bit. Since the XNOR is 1 if both input bits are the same and 0 if they are not, Avida refers to the XNOR as an EQU (Lenski et al., 2003).

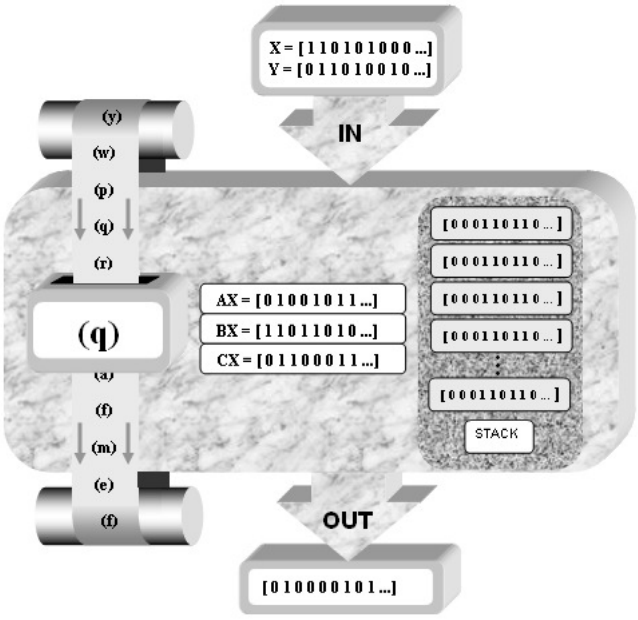

Figure 4.1: The digital organism used by Avida, similar to that shown in the original Avida paper (Lenski et al., 2003).

Generating a program from a fixed alphabet of 26 instructions (see Table 4.1) to perform a specified function like XNOR (EQU) is similar to generating a specified word or phrase from the English alphabet somewhere within an alphabet of characters. The Avida simulation actually make use of three inputs, X, Y and Z, in a circular queue. Whenever an input is read by the program, it is removed from the queue and then placed at the back of the queue. As a result, three consecutive reads will read in three different inputs, whereas the fourth will read the first input in again. This property was not addressed in the original paper.

Avida uses a small alphabet of instructions (see Table 4.1) to import the inputs, perform manipulation, and export the output. The instruction tape runs in a continuous loop. The number of entries can change during the search process. Each letter in the loop in Figure 4.1 corresponds to the lettered instructions in Table 4.1. The highlighted instruction (q) `IO`, for example, outputs the target register, checks

to see if any logic function in Table 4.2 has been performed, and reads the next input into the target register. Each of the registers in the Avida contains 32 bits although, as we will see, the number of bits in each register does not significantly impact the search. The inputs X and Y are assigned randomly, are read only, and forever remain fixed. There are read-write scratch pad registers in the organism, AX, BX, and CX, on which to perform the operations. There is also a stack that can pop a stored 32 bit word into AX, BX and CX. An element from AX, BX and CX can likewise be pushed onto the top of the stack. The goal is to choose instructions from Table 4.1 so that the output, written by operation (q) `IO`, is the bitwise XNOR of the input registers X and Y. For most generations of the evolutionary search, Avida uses 3600 of the digital organisms in Figure 4.1.

4.1 Problem Difficulty

Like choosing letters from the alphabet to form a sequence of letters that will pass a spell check, the goal of Avida is to search for a sequence of instructions that has meaning with respect to the logic functions in Table 4.2. The question is - how difficult is it to generate the logic functions in Table 4.2 using the $N = 26$ instructions in Table 4.1 in a sequence of instructions?

4.2 Instruction Count

We want to evaluate the relative performance between Avida and competing algorithms. The most straightforward measure of computational cost is measurement of the actual CPU time of the process. CPU time, however, can differ considerably on different hardware. Other external factors, such as user interaction with the system, may also result in a non-accurate measure of the computational cost.

Another measure of the cost in search algorithms is a query count. In Avida, however, the computational demands of a query can vary widely.

Table 4.1: $N = 26$ instructions for performing the logic functions in Table 4.2.

#	Operation	Description
(a)	`nop-A`	No-Operation. May modify the operation of the instruction before it.
(b)	`nop-B`	No-Operation. May modify the operation of the instruction before it.
(c)	`nop-C`	No-Operation. May modify the operation of the instruction before it.
(d)	`if-n-equal`	Compares two values, skips the next instruction if they are equal.
(e)	`if-less`	Compares two values, skips the next instruction if the first is less.
(f)	`push`	Puts a copy of the value in the target register onto the top of the stack.
(g)	`pop`	Removes a value from the top of the stack and places it into the target register.
(h)	`swap-stk`	Switches the stack between two available stacks
(i)	`swap`	Swaps the value of the target register with the next register
(j)	`shift-r`	Shift the bits of the value in the target register to the right. This is the same as dividing by two and rounding down.
(k)	`shift-l`	Shift the bits of the value in the target register to the left. This is the same as multiplying by two.
(l)	`inc`	Increments the value in the target register by one.
(m)	`dec`	Decrements the value in the target register by one.
(n)	`add`	Adds the values of the BX and CX register, placing the result in the target register.
(o)	`sub`	Subtract the value of CX from BX, placing the result in the target register.
(p)	`nand`	Bitwise-nands the values of BX and CX together, placing the result in the target register.
(q)	`IO`	Outputs the target register, checking if for any tasks completed. Reads the next input into the target register.
(r)	`h-alloc`	Allocates additional memory to be used for a daughter organism.
(s)	`h-divide`	Splits the daughter and the parent into seperate organisms.
(t)	`h-copy`	Copies a single instruction from the read head to the write head
(u)	`h-search`	Searches the genome for a pattern of nops, moving the flow-head to their location.
(v)	`mov-head`	Moves one of the other heads to the flow-head.
(w)	`jmp-head`	Causes one of the heads the jump forward by the value in the CX register.
(x)	`get-head`	Copies the position of one of the heads into the CX register.
(y)	`if-label`	Checks to see whether a certain pattern of nops has just been copied. If they have not, skips an instruction.
(z)	`set-flow`	Sets the position of the flow head to the value in target register.

Table 4.2: NAND logic illustrated in Figures 4.2 through 4.4

The "&" denotes a logic AND, the "+" is an OR and the overline denotes complement. In the three "either-or" functions, a success is claimed if either one of the two functions is performed. The number in the "nands" column is the $\log_2$ of the "Score" column.

Logic	**OUT**	**NANDS**	**Score**
NOT	either $\overline{\text{X}}$ or $\overline{\text{Y}}$	1	2
NAND	$\overline{\text{X\&Y}}$	1	2
AND	X&Y	2	4
OR_N	either X+$\overline{\text{Y}}$ or $\overline{\text{X}}$+Y	2	4
OR	X+Y	3	8
AND_N	either X&$\overline{\text{Y}}$ or $\overline{\text{X}}$&Y	3	8
NOR	$\overline{\text{X + Y}}$	4	16
XOR	X$\oplus$Y=(X&$\overline{\text{Y}}$)+($\overline{\text{X}}$&Y)	4	16
XNOR	$\overline{\text{X} \oplus \text{Y}}$=(X&Y)+($\overline{\text{X}}$&$\overline{\text{Y}}$)	5	32

A reasonable accounting metric for Avida is an instruction count. We define an instruction as the execution of any one of the entries in Table 4.1.

4.3 A Single Avida Organism

Avida's performance can be evaluated using Monte Carlo simulation. Here are some useful definitions.

- *A program* is a list of 100 instructions. 85 are chosen randomly and 15 are specified.[2] The number of instructions in an executed program varies. The same instruction, for example, can be executed more than once.
- *A query* is a sequence of programs executed until either an XNOR is found, or until

$$\acute{I} = 10.8 \text{ billion instructions} \tag{4.1}$$

[2] These 15 instructions, native to the Avida digital organism, allow the process of replication in the evolutionary search.

are used. This number is roughly the number of instructions used by the Avida default.[3]

- A program or query is *diminished* if instructions (v) `mov-head` and (w) `jmp-head` in Table 4.1 are not used. They nearly always cause the failure of an XNOR calculation.[4] A program or query including all of the instructions, including (v) `mov-head` and (w) `jmp-head`, is dubbed *full*.
- Let D denote *diminished* and F *full*. P_D and P_F denote diminished and full programs while Q_D and Q_F correspond to diminished and full queries. An S will denote success in finding an XNOR, so $P_{S|F}$ and $Q_{S|F}$ denote successful full programs and queries, while $P_{S|D}$ and $Q_{S|D}$ are the diminished equivalents.

4.4 An Estimate of the Endogenous Information of a Full Program

A total of $|Q_D| = 1420$ diminished queries were performed and a total of $|S| = 21$ XNOR's resulted. The average number of instructions per diminished program, P_D, was

$$\mathbb{E}[I \text{ per } P_D] \simeq 637.9 \text{ instructions} \tag{4.2}$$

so the total number of diminished programs simulated is

$$|P_D| = \frac{\acute{I}\,|Q_D|}{E[I \text{ per } P_D]} \simeq 2.40 \times 10^{10}$$

A separate run of full programs[5] gave

$$\mathbb{E}[I \text{ per } P_F] \simeq 1972 \text{ instructions.} \tag{4.3}$$

[3] $\acute{I}$ = 100,000 updates × 3600 programs × 30 instructions (on average) per program per update (Lenski et al., 2003).

[4] The instructions (v) `mov-head` and (w) `jmp-head` both manipulate various heads in the Avida CPU. There are three heads which can be manipulated by these instruction, the read-head, write-head, or the instruction pointer. The read and write heads are setup for the copying process at the beginning of an Avida program. If they are moved, the replication process will not work correctly. If the instruction pointer is changed the Avida program will jump to a different position in its program, in all probability causing it to loop in a particular portion of its program never reaching the copy loop.

[5] 10,272,633 full programs

The maximum number of instructions allowed by Avida was 2000. Of the $N_F = 26^{85} = 1.87 \times 10^{120}$ possible full programs, there are over 10^{108} that compute an XNOR. To show this, consider first the number of diminished programs. $N_D = 24^{85} = 2.08 \times 10^{117}$. The probability a diminished program will compute an XNOR is about $\Pr[\text{XNOR}|P_D] \simeq |S|/|P_D| = 8.73.5 \times 10^{-10}$. The number of diminished programs capable of computing XNOR is therefore $|P_{S|D}| = N_D \times \Pr[\text{XNOR}|P_D] \simeq 1.82 \times 10^{108}$. If we assume all programs containing (v) `mov-head` or (w) `jmp-head` will fail, this is also the number of full programs that will compute XNOR, *i.e.* $|P_{S|F}| \simeq |P_{S|D}|$. The probability of choosing a successful program randomly from the set of full programs is thus

$$p = \Pr[P_{S|F}] \simeq \frac{|P_{S|D}|}{N_F} = 9.69 \times 10^{-13}. \tag{4.4}$$

The endogenous information of the difficulty of generating an XNOR with 85 randomly selected instructions from Table 4.1 follows from (1.1) as

$$I_\Omega \simeq 40 \text{ bits}$$

4.5 Algorithm $\mathfrak{Q}$, Repeated Diminished Programs

Running a program a repeated number of times generates more information than a single run (Dembski and Marks, 2009b). For the $|Q_D| = 1420$ diminished query simulations each to a maximum of $\acute{I}$ instructions each, the NAIPI, calculated using (**??**), was

$$I_\oplus / I_\Omega \simeq 5.78 \times 10^{-12} \tag{4.5}$$

This is the (small) NAIPI obtained from (a) repeated queries and (b) using only diminished programs.

4.6 Stair Step Active Information in the NAND Search

Knowledge that higher level nand logic can be built on lower level nand logic can be used to introduce *stair step active information* into the search for the XNOR.

Stair step active information, discussed in detail elsewhere (Dembski and Marks, 2009b), is an important source of active information in Avida.

Examples of synthesizing the nine logic functions in Table 4.2 using only nand gates is well known and is illustrated in minimal form[6] in Figures 4.2 through 4.4. They are included here to show that the logic can build on itself. For example, Figure 4.2 logic function #3 shows the AND as simply a cascade connection of the NOT circuit #1 and the NAND function in #2. Higher up the list, the #9 XNOR (EQU) in Figure 4.4 is a simple cascade connection of the #8 XOR circuit with the #2 NAND. The #8 XOR, in turn, can be synthesized using other lower numbered logic circuits. Not all of the steps need be directly useful to the subsequent target.

In the nand gate synthesis, logic functions with fewer gates are rewarded as possible steps towards achieving the ultimate XNOR target at the top of the stairs. The rewarding in Avida is done using the Score column in Table 4.2. The $\log_2$ of the Score is equal to the minimum number of gates required to perform the logic function. If two or more logic functions are generated by a digital organism, the fitness of the organism is the product of the scores. Each logic function is counted only once in this tally. For example, if an organism generates three separate cases of the NOT function, the fitness rule only counts them as one occurrence.

For all of the subsequent simulations, a resource bound of $\acute{I}$ in (4.1) is imposed.

Using 3600 digital organisms of the type shown in Figure 4.1, Avida weighs the fitness of an organism from the Score given in Table 4.2. A detailed description of other operations used in Avida, including mutation rates and replication properties, are available elsewhere (Lenski et al., 2003).

- *Algorithm* $\mathcal{A}_1$. Results using Avida's default values, including all of the stair steps in Table 4.2 and all $N = 26$ instructions in Table 4.2, are shown in the first row of Table 4.3. Most of the trials resulted in a success. The NAIPI for

[6] Figures 4.2 through 4.4 show configurations using the fewest possible number of nand gates. Other configurations are possible.

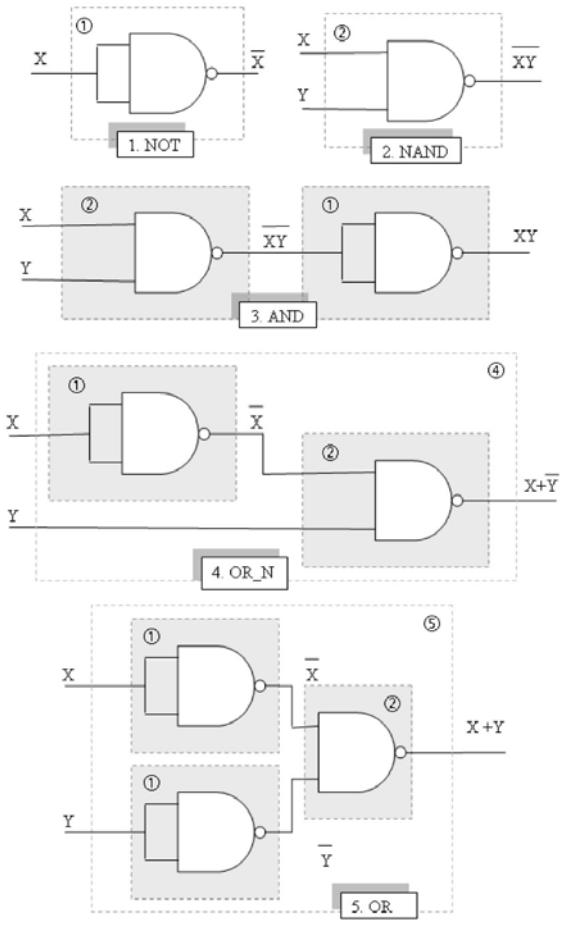

Figure 4.2: Minimal NAND logic. Logic function is synthesized using the minimum possible number of NAND gates. Continued in Figure 4.3.

$\mathcal{A}_1$ is $I_\oplus/I_\Omega = 1.9 \times 10^{-9}$. This value is due, in part, to active information introduced by the stair steps.

- *Algorithm* $\mathcal{A}_2$. As is shown in the second line in Table 4.3, the NAIPI is reduced by excluding the XOR and NOR steps. Removing these steps therefore makes the search more difficult.[7]
- *Algorithms* $\mathcal{A}_3$ *and* $\mathcal{A}_4$. Taking away additional stair steps worsens the performance even more. In $\mathcal{A}_3$, the additional functions of OR and AND_N

[7] There are searches where removing steps can actually improve performance (Dembski and Marks, 2010b).

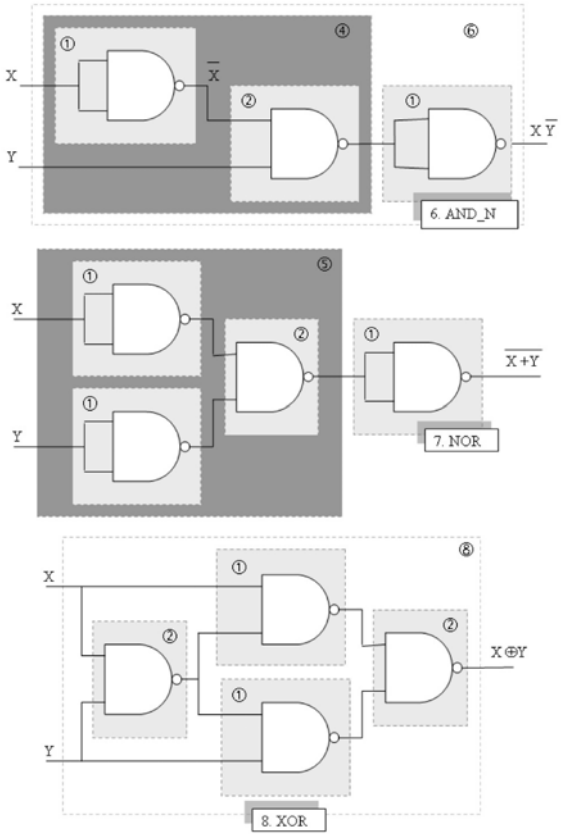

Figure 4.3: NAND logic. Continued from Figure 4.2.

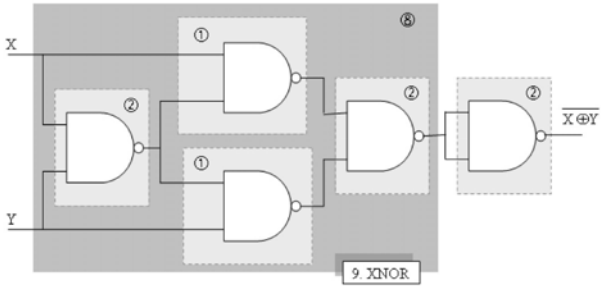

Figure 4.4: NAND logic for the XNOR operation. Continued from Figure 4.3.

are removed resulting in an NAIPI of less than half when compared to using all of the stair steps. If, instead, AND and OR_N are removed, the NAIPI reduction is over two thirds. These stair steps therefore contribute significantly to the active information of the search.

Table 4.3: The effects of removing stair steps from Avida ($\mathring{A}$)and the ratcheted evolutionary strategy searches ($\mathcal{R}$) for the XNOR.

The estimate of NAIPI is computed using (??). The comparative numbers for repeated running of randomly generated diminished Avida programs, as described in Section 4.5, is in the last row. All simulations, except $\mathfrak{Q}$, have 353 runs. The run count for $\mathfrak{Q}$ is 1420 queries. The values in the $I_{\oplus}/I_{\Omega}$ column should be multiplied by 10^{-9}.

Model	Steps Removed	Successes	$I_{\oplus}/I_{\Omega}$
$\mathcal{A}_1$	None (All Steps Enabled)	346	1.90
$\mathcal{A}_2$	XOR/NOR	319	1.37
$\mathcal{A}_3$	XOR/NOR/OR/AND_N	227	0.62
$\mathcal{A}_4$	XOR/NOR/AND/OR_N	222	0.52
$\mathcal{R}_1$	None (All Steps Enabled)	353	25.30
$\mathcal{R}_2$	XOR/NOR	353	16.26
$\mathcal{R}_3$	XOR/NOR/OR/AND_N	353	6.72
$\mathcal{R}_4$	XOR/NOR/AND/OR_N	353	8.05
$\mathfrak{Q}$	None (Random)	21	5.78×10^{-3}

Available knowledge for a given search problem can be used with varying efficiency to produce active information. The stair step knowledge available to Avida is a rich source of active information and, in terms of efficient search, is used poorly by Avida in the search for XNOR. We consider an alternate ratchet search strategy using only a single digital organism. A generation in the search consists of replacing an instruction in the loop by a randomly chosen instruction. If the fitness of the organism does not decrease, we keep the mutation and repeat the iteration. If the fitness does increase, the mutation is discarded and the process repeated. Different versions of this procedure are shown in Table 4.3 and are labeled $\mathcal{R}_1$ through $\mathcal{R}_4$. Each uses the same stair steps as algorithms $\mathcal{A}_1$ through $\mathcal{A}_4$. The same instruction bound and trial number are used in the ratchet simulations. $\mathcal{R}_1$ uses the same resources as $\mathcal{A}_1$ and increases the NAIPI over an order of magnitude. The NAIPI for $\mathcal{R}_2$ through $\mathcal{R}_4$ are likewise bettered significantly with respect to their corresponding Avida implementations in $\mathcal{A}_2$ through $\mathcal{A}_4$. As was the case with the Avida program,

removing steps in the ratchet approach decreases algorithm performance as measured by active information.

Will Avida work without the stair step active information being hard wired into the process? The Avida paper reports that, in all case studies using stair step active information (Lenski et al., 2003),

> "... at least one population evolved EQU."

EQU is the same as XNOR. What happens when no stair step active information is applied?

> "At the other extreme, 50 populations evolved in an environment where only EQU was rewarded, and no simpler function yielded energy. We expected that EQU would evolve much less often because selection would not preserve the simpler functions that provide foundations to build more complex features. Indeed, none of these populations evolved EQU, a highly significant difference from the fraction that did so in the reward-all environment." (Lenski et al., 2003)

Therefore, hard wired stair step active information is essential in order for Avida to produce results in reasonable time. We were able to do so in Section 4.3 only because the instruction set was diminished and the query count far exceeded the effort reported in the Avida paper (Lenski et al., 2003).

4.7 Instruction Selection Active Information in the NAND Search

Some of the instructions in Table 4.1 are essential to do any of the logic functions in Table 4.2. Performing any logic function without the (p) `nand` instruction, for example, would be difficult. Conversely, there are instructions in Table 4.1 that make little or no contributions to the goals of Avida. The instructions (j) `shift-r`, (k) `shift-l`, (l) `inc`, (m) `dec` (n) `add`, and (o) `sub`, for example, are typically deleterious to the performance of a logic operation.

The effects of removing various sets of instructions in Avida and the ratcheted evolutionary strategy searches are shown in Table 4.4. For both Avida and the ratchet

Table 4.4: Effects of instruction removal on Avida ($\mathcal{A}$) and the ratcheted evolutionary strategy searches ($\mathcal{R}$) for XNOR.

The letters in the Instructions column correspond to the letter labeling of the instructions in Table 4.1. The entries for $\mathcal{A}_1$ and $\mathcal{R}_1$ in Table 4.3 are repeated here for ease of comparison. Ratchet algorithms $\mathcal{R}_5$ and $\mathcal{R}_6$ used the same instructions as $\mathcal{A}_5$ and $\mathcal{A}_6$. All simulations have 353 runs. The values in the $I_\oplus/I_\Omega$ column should be multiplied by 10^{-9}.

Model	Instructions	Successes	$I_\oplus/I_\Omega$
$\mathcal{A}_1$	`abcdefghijklmnopqrstuvwxyz`	346	1.90
$\mathcal{A}_5$	`abc--fghi--lmnopqrstuv--y-`	353	5.16
$\mathcal{A}_6$	`abc--fg-i------pqrstuv--y-`	353	10.00
$\mathcal{R}_1$	`abcdefghijklmnopqrstuvwxyz`	353	25.30
$\mathcal{R}_5$	`abc--fghi--lmnopqrstuv--y-`	353	197.76
$\mathcal{R}_6$	`abc--fg-i------pqrstuv--y-`	353	593.80

search, removal of deleterious or otherwise nonessential instructions increases the active information significantly. Using only 14 of the 26 instructions in $\mathcal{R}_6$ increases the NAIPI with respect to the vanilla Avida in $\mathcal{A}_1$ by a factor of over 300.

4.8 Initialization Active Information in the NAND Search

Another potential source of active information in search algorithms is initialization. If a search can be initialized in some sense closer to a solution, then we might expect faster convergence. Like any source of active information, the prior knowledge must be right. If we place the initialization farther from a solution, for example, convergence can take much longer.

In Avida, the instruction (c) `nop-C` plays an important role. The BX register is used as a default register and is used unless changed by a (a) `nop-A` or or (c) `nop-C` instruction. In the default operation of nand, for example, the nand operation of registers BX and CX is performed and deposited in register BX. Storing intermediate values of Avida's computation in register CX is therefore mandatory in computing the nand and can only be done using the (c) `nop-C` instruction. Apparently aware of this requirement, the designer of Avida set the initial value of all instructions

Table 4.5: Effects of different initializations on the Avida ($\mathcal{A}$) for XNOR.

The DOA designation indicates that nearly all organisms died before producing any offspring. In $\mathcal{A}_9$, deaths were caused largely by instructions (v) `mov-head` and (w) `jmp-head`. These entries were removed in $\mathcal{A}_{11}$ and successes were achieved. A random selection of `nop-A`, `nop-B`, `nop-C` in algorithm $\mathcal{A}_{10}$ also resulted in a DOA scenario. [The reason is as follows. As part of the replication loop, Avida searches for patterns of `nop`'s. In particular the end of the Avida program is marked by `nop-A`, `nop-B`. If that pattern appears anywhere else in the program it will be detected as the end of the program thus causing the replicator to fail. With 85 randomly chosen `nop`'s, that pattern is almost certain to appear.] The entry for $\mathcal{A}_1$ from Table 4.3 is repeated here for ease of comparison. All simulations have 353 runs. The values in the $I_\oplus/I_\Omega$ column should be multiplied by 10^{-9}.

Model	Initialization	Successes	$I_\oplus/I_\Omega$
$\mathcal{A}_1$	`nop-C`	346	1.90
$\mathcal{A}_7$	`nop-B`	318	0.81
$\mathcal{A}_8$	`nop-A`	324	0.85
$\mathcal{A}_9$	Random	DOA	DOA
$\mathcal{A}_{10}$	Random-`nops`	DOA	DOA
$\mathcal{A}_{11}$	Random w/o (u) and (v)	274	0.84

not dedicated to organism replication to `nop-C`. The performance of Avida with this initialization is superior to initialization using nonmandatory `nop-A` or `nop-B` instructions. The performance of these and other initializations are summarized in Table 4.5. In all cases, the NAIPI deteriorates by at least a factor of a half, or as described in the figure caption, is DOA.

4.9 Active Information

The Avida program uses numerous sources of active information to guide its performance to successful discovery of the XNOR logic function. The sources include the following.

- *Stair step active information.* In the initial description of Avida, the authors write (Lenski et al., 2003)

> "Some readers might suggest that we stacked the deck by studying the evolution of a complex feature that could be built on simpler functions that were also useful."

This, indeed, is what the writers of Avida software do when using stair step active information. The importance of stair step active information is evident from the inability to generate a single XNOR in Avida without using it (Lenski et al., 2003).

- *Active information from Avida's initialization.* The initialization in Avida recognizes the essential role of the `nop-C` instruction in finding the XNOR. Initializing using all nonessential `nop-A` or `nop-B` instructions results in the a decrease in NAIPI in Avida.
- *Mutation, fitness, and choosing the fittest of a number of mutated offspring* (Dembski and Marks, 2009b) are additional sources of active information in Avida we have not explored in this paper.

CHAPTER FIVE

Conclusions

The success of a search algorithm depends on the problem specific sources of knowledge that are used during the search process. The case studies considered in this thesis all demonstrate the importance of problem specific information being present in the search algorithm. That is, the search algorithm has to be designed to solve particular problems. Additionally, for each of the search problems considered, alternative algorithms exist which perform better then the "traditional" algorithm for that problem.

In the case of the hamming distance search, we have shown that there are a variety of algorithms with different degrees of efficiency. In particular, the FOOHOA algorithm takes particular advantage of the characteristics of the hamming oracle to provide a very efficient search process. This shows that the hamming oracle is a good source of knowledge enabling the search algorithms to locate the target rather then any unique properties of the search algorithms themselves.

Ev uses a hamming oracle much like the first case. However, the use of the perceptron struture makes the search problem much easier then it would be otherwise. Essentially, Ev is biased towards producing outputs like those that are the targets of its search. This constitutes active information in the search process assisting it.

Avida makes use of a stairstep reward system whereby simpler logic functions are rewarded. Since more complex logic functions are built on simpler logic functions, this provides a great deal of assistance in the development of the complex logic functions. Additionally, Avida makes us of an initialization that provides "hints" as to what needs to happen in the final program. As such, Avida makes use of problem specific information in order to produce its binary logic functions.

All of these algorithms make use of problem specific knowledge. This supports the contention of the various conservation of information theoreoms that solving search problems requires the exploitation of problem specific knowledge about the search problem.

APPENDICES

APPENDIX A

Copyrights

IEEE COPYRIGHT AND CONSENT FORM

To ensure uniformity of treatment among all contributors, other forms may not be substituted for this form, nor may any wording of the form be changed. This form is intended for original material submitted to the IEEE and must accompany any such material in order to be published by the IEEE.Please read the form carefully and keep a copy for your files.

TITLE OF PAPER/ARTICLE/REPORT, INCLUDING ALL CONTENT IN ANY FORM, FORMAT, OR MEDIA (hereinafter, "The Work"):**Evolutionary Synthesis of Nand Logic: Dissecting a Digital Organism**

COMPLETE LIST OF AUTHORS:**WinstonJ Ewert and Bob Marks II**

IEEE PUBLICATION TITLE (Journal, Magazine, Conference, Book):**2009 IEEE International Conference on Systems, Man& Cybernetics**

COPYRIGHT TRANSFER

1. The undersigned hereby assigns to The Institute of Electrical and Electronics Engineers, Incorporated (the IEEE) all rights under copyright that may exist in and to: (a) the above Work, including any revised or expanded derivative works submitted to the IEEE by the undersigned based on the Work; and (b) any associated written or multimedia components or other enhancements accompanying the Work.

See Retained Rights ,below.

CONSENT AND RELEASE

2. In the event the undersigned makes a presentation based upon the Work at a conference hosted or sponsored in whole or in part by the IEEE, the undersigned, in consideration for his/her participation in the conference, hereby grants the IEEE the unlimited, worldwide, irrevocable permission to use, distribute, publish, license, exhibit, record, digitize, broadcast, reproduce and archive, in any format or medium, whether now known or hereafter developed: (a) his/her presentation and comments at the conference; (b) any written materials or multimedia files used in connection with his/her presentation; and (c) any recorded interviews of him/her (collectively, the Presentation). The permission granted includes the transcription and reproduction of the Presentation for inclusion in products sold or distributed by IEEE and live or recorded broadcast of the Presentation during or after the conference.

3. In connection with the permission granted in Section 2, the undersigned hereby grants IEEE the unlimited, worldwide, irrevocable right to use his/her name, picture, likeness, voice and biographical information as part of the advertisement, distribution and sale of products incorporating the Work or Presentation, and releases IEEE from any claim based on right of privacy or publicity.

4. The undersigned hereby warrants that the Work and Presentation (collectively, the Materials) are original and that he/she is the author of the Materials. To the extent the Materials incorporate text passages, figures, data or other material from the works of others, the undersigned has obtained any necessary permissions. Where necessary, the undersigned has obtained all third party permissions and consents to grant the license above and has provided copies of such permissions and consents to IEEE.

[ ] Please check this box if you do not wish to have video/audio recordings made of your conference presentation.

AUTHOR RESPONSIBILITIES

The IEEE distributes its technical publications throughout the world and wants to ensure that the material submitted to its publications is properly available to the readership of those publications. Authors must ensure that their Work meets the requirements as stated in section 8.2.1 of the IEEE PSPB Operations Manual, including provisions covering originality, authorship, author responsibilities and author misconduct. More information on IEEE's publishing policies may be found at http://www.ieee.org/web/publications/pubtoolsandpolicyinfo/index.html. Authors are advised especially of IEEE PSPB Operations Manual section 8.2.1.B12: "It is the responsibility of the authors, not the IEEE, to determine whether disclosure of their material requires the prior consent of other parties and, if so, to obtain it." Authors are also advised of IEEE PSPB Operations Manual section 8.1.1B: "Statements and opinions given in work published by the IEEE are the expression of the authors."

RETAINED RIGHTS/TERMS AND CONDITIONS

1. Authors/employers retain all proprietary rights in any process, procedure, or article of manufacture described in the Work.
2. Authors/employers may reproduce or authorize others to reproduce The Work, material extracted verbatim from the Work, or derivative works to the extent permissible under United States law for works authored by U.S. Government employees, and for the author's personal use or for company or organizational use, provided that the source and any IEEE copyright notice are indicated, the copies are not used in any way that implies IEEE endorsement of a product or service of any employer, and the copies themselves are not offered for sale.
3. Authors/employers may make limited distribution of all or portions of the Work prior to publication if they inform the IEEE in advance of the nature and extent of such limited distribution.
4. In the case of a Work performed under a U.S. Government contract or grant, the IEEE recognizes that the U.S. Government has royalty-free permission to reproduce all or portions of the Work, and to authorize others to do so, for official U.S. Government purposes only, if the contract/grant so requires.
5. For all uses not covered by items 2, 3, and 4, authors/employers must request permission from the IEEE Intellectual Property Rights office to reproduce or authorize the reproduction of the Work or material extracted verbatim from the Work, including figures and tables.
6. Although authors are permitted to re-use all or portions of the Work in other works, this does not include granting third-party requests for reprinting, republishing, or other types of re-use. The IEEE Intellectual Property Rights office must handle all such third-party requests.

INFORMATION FOR AUTHORS

IEEE Copyright Ownership

It is the formal policy of the IEEE to own the copyrights to all copyrightable material in its technical publications and to the individual contributions contained therein, in order to protect the interests of the IEEE, its authors and their employers, and, at the same time, to facilitate the appropriate re-use of this material by others. The IEEE distributes its technical publications throughout the world and does so by various means such as hard copy, microfiche, microfilm, and electronic media. It also abstracts and may translate its publications, and articles contained therein, for inclusion in various compendiums, collective works, databases and similar publications.

Author/Employer Rights

If you are employed and prepared the Work on a subject within the scope of your employment, the copyright in the Work belongs to your employer as a work-for-hire. In that case, the IEEE assumes that when you sign this Form, you are authorized to do so by your employer and that your employer has consented to the transfer of copyright, to the representation and warranty of publication rights, and to all other terms and conditions of this Form. If such authorization and consent has not been given to you, an authorized representative of your employer should sign this Form as the Author.

Reprint/Republication Policy

The IEEE requires that the consent of the first-named author and employer be sought as a condition to granting reprint or republication rights to others or for permitting use of a Work for promotion or marketing purposes.

GENERAL TERMS

1. The undersigned represents that he/she has the power and authority to make and execute this assignment.
2. The undersigned agrees to indemnify and hold harmless the IEEE from any damage or expense that may arise in the event of a breach of any of the warranties set forth above.
3. In the event the above work is not accepted and published by the IEEE or is withdrawn by the author(s) before acceptance by the IEEE, the foregoing copyright transfer shall become null and void and all materials embodying the Work submitted to the IEEE will be destroyed.
4. For jointly authored Works, all joint authors should sign, or one of the authors should sign as authorized agent for the others.

Winston Ewert
Author/Authorized Agent For Joint Authors

29-06-2009
Date(dd-mm-yy)

IEEE COPYRIGHT AND CONSENT FORM

To ensure uniformity of treatment among all contributors, other forms may not be substituted for this form, nor may any wording of the form be changed. This form is intended for original material submitted to the IEEE and must accompany any such material in order to be published by the IEEE.Please read the form carefully and keep a copy for your files.

TITLE OF PAPER/ARTICLE/REPORT, INCLUDING ALL CONTENT IN ANY FORM, FORMAT, OR MEDIA (hereinafter, "The Work"):**Efficient Per Query Information Etraction from a Hamming Oracle**

COMPLETE LIST OF AUTHORS:**Winston Ewert, George Montaez, Robert J. Marks II**

IEEE PUBLICATION TITLE (Journal, Magazine, Conference, Book): **2nd Meeting of the Southeastern Symposium on System Theory**

COPYRIGHT TRANSFER

1. The undersigned hereby assigns to The Institute of Electrical and Electronics Engineers, Incorporated (the IEEE) all rights under copyright that may exist in and to: (a) the above Work, including any revised or expanded derivative works submitted to the IEEE by the undersigned based on the Work; and (b) any associated written or multimedia components or other enhancements accompanying the Work.

See Retained Rights ,below.

CONSENT AND RELEASE

2. In the event the undersigned makes a presentation based upon the Work at a conference hosted or sponsored in whole or in part by the IEEE, the undersigned, in consideration for his/her participation in the conference, hereby grants the IEEE the unlimited, worldwide, irrevocable permission to use, distribute, publish, license, exhibit, record, digitize, broadcast, reproduce and archive, in any format or medium, whether now known or hereafter developed: (a) his/her presentation and comments at the conference; (b) any written materials or multimedia files used in connection with his/her presentation; and (c) any recorded interviews of him/her (collectively, the Presentation). The permission granted includes the transcription and reproduction of the Presentation for inclusion in products sold or distributed by IEEE and live or recorded broadcast of the Presentation during or after the conference.

3. In connection with the permission granted in Section 2, the undersigned hereby grants IEEE the unlimited, worldwide, irrevocable right to use his/her name, picture, likeness, voice and biographical information as part of the advertisement, distribution and sale of products incorporating the Work or Presentation, and releases IEEE from any claim based on right of privacy or publicity.

4. The undersigned hereby warrants that the Work and Presentation (collectively, the Materials) are original and that he/she is the author of the Materials. To the extent the Materials incorporate text passages, figures, data or other material from the works of others, the undersigned has obtained any necessary permissions. Where necessary, the undersigned has obtained all third party permissions and consents to grant the license above and has provided copies of such permissions and consents to IEEE.

[ ] Please check this box if you do not wish to have video/audio recordings made of your conference presentation.

AUTHOR RESPONSIBILITIES

The IEEE distributes its technical publications throughout the world and wants to ensure that the material submitted to its publications is properly available to the readership of those publications. Authors must ensure that their Work meets the requirements as stated in section 8.2.1 of the IEEE PSPB Operations Manual, including provisions covering originality, authorship, author responsibilities and author misconduct. More information on IEEE's publishing policies may be found at http://www.ieee.org/web/publications/pubtoolsandpolicyinfo/index.html. Authors are advised especially of IEEE PSPB Operations Manual section 8.2.1.B12: "It is the responsibility of the authors, not the IEEE, to determine whether disclosure of their material requires the prior consent of other parties and, if so, to obtain it." Authors are also advised of IEEE PSPB Operations Manual section 8.1.1B: "Statements and opinions given in work published by the IEEE are the expression of the authors."

RETAINED RIGHTS/TERMS AND CONDITIONS

1. Authors/employers retain all proprietary rights in any process, procedure, or article of manufacture described in the Work.
2. Authors/employers may reproduce or authorize others to reproduce The Work, material extracted verbatim from the Work, or derivative works to the extent permissible under United States law for works authored by U.S. Government employees, and for the author's personal use or for company or organizational use, provided that the source and any IEEE copyright notice are indicated, the copies are not used in any way that implies IEEE endorsement of a product or service of any employer, and the copies themselves are not offered for sale.
3. Authors/employers may make limited distribution of all or portions of the Work prior to publication if they inform the IEEE in advance of the nature and extent of such limited distribution.
4. In the case of a Work performed under a U.S. Government contract or grant, the IEEE recognizes that the U.S. Government has royalty-free permission to reproduce all or portions of the Work, and to authorize others to do so, for official U.S. Government purposes only, if the contract/grant so requires.
5. For all uses not covered by items 2, 3, and 4, authors/employers must request permission from the IEEE Intellectual Property Rights office to reproduce or authorize the reproduction of the Work or material extracted verbatim from the Work, including figures and tables.
6. Although authors are permitted to re-use all or portions of the Work in other works, this does not include granting third-party requests for reprinting, republishing, or other types of re-use. The IEEE Intellectual Property Rights office must handle all such third-party requests.

INFORMATION FOR AUTHORS

IEEE Copyright Ownership

It is the formal policy of the IEEE to own the copyrights to all copyrightable material in its technical publications and to the individual contributions contained therein, in order to protect the interests of the IEEE, its authors and their employers, and, at the same time, to facilitate the appropriate re-use of this material by others. The IEEE distributes its technical publications throughout the world and does so by various means such as hard copy, microfiche, microfilm, and electronic media. It also abstracts and may translate its publications, and articles contained therein, for inclusion in various compendiums, collective works, databases and similar publications.

Author/Employer Rights

If you are employed and prepared the Work on a subject within the scope of your employment, the copyright in the Work belongs to your employer as a work-for-hire. In that case, the IEEE assumes that when you sign this Form, you are authorized to do so by your employer and that your employer has consented to the transfer of copyright, to the representation and warranty of publication rights, and to all other terms and conditions of this Form. If such authorization and consent has not been given to you, an authorized representative of your employer should sign this Form as the Author.

Reprint/Republication Policy

The IEEE requires that the consent of the first-named author and employer be sought as a condition to granting reprint or republication rights to others or for permitting use of a Work for promotion or marketing purposes.

GENERAL TERMS

1. The undersigned represents that he/she has the power and authority to make and execute this assignment.
2. The undersigned agrees to indemnify and hold harmless the IEEE from any damage or expense that may arise in the event of a breach of any of the warranties set forth above.
3. In the event the above work is not accepted and published by the IEEE or is withdrawn by the author(s) before acceptance by the IEEE, the foregoing copyright transfer shall become null and void and all materials embodying the Work submitted to the IEEE will be destroyed.
4. For jointly authored Works, all joint authors should sign, or one of the authors should sign as authorized agent for the others.

Winston Ewert — **29-06-2009**

Author/Authorized Agent For Joint Authors — **Date(dd-mm-yy)**

BIBLIOGRAPHY

Bäck, Thomas. *Evolutionary Algorithms in Theory and Practice*. Oxford Oxfordshire: Oxford University Press, 1996.

Bäck, Thomas, and Hans-Paul Schwefel. "An overview of evolutionary algorithms for parameter optimization." *Evol. Comput.* 1, 1: (1993) 1–23.

Brillouin, Leon. *Science and Information Theory*. New York: Academic Press, Inc., 1962, second edition.

Chan, Pak. *Digital Design Using Field Programmable Gate Arrays*. Englewood Cliffs: PTR Prentice Hall, 1994.

Cover, T. *Elements of Information Theory*. New York: Wiley, 1991.

Dembski, William A., and Robert J. Marks. "Bernoulli's principle of insufficient reason and conservation of information in computer search." In *SMC'09: Proceedings of the 2009 IEEE international conference on Systems, Man and Cybernetics*. Piscataway, NJ, USA: IEEE Press, 2009a, 2647–2652.

———. "Conservation of information in search: measuring the cost of success." *Trans. Sys. Man Cyber. Part A* 39, 5: (2009b) 1051–1061.

———. *The Nature of Nature*, Wilmington, Del.: ISI Books, 2010a, chapter Life's Conservation Law: Why Darwinian Evolution Cannot Create Biological Information.

———. "The Search for a Search: Measuring the Information Cost of Higher Level Search." *Journal of Advanced Computational Intelligence and Intelligent Informatics* 14, 5: (2010b) 475–486.

El Gamal, A., JL Kouloheris, D. How, and M. Morf. "BiNMOS: a basic cell for BiCMOS sea-of-gates." In *Custom Integrated Circuits Conference, 1989., Proceedings of the IEEE 1989*. 1989, 831–834.

English, Thomas M. "Some Information Theoretic Results on Evolutionary Optimization." In *Proceedings of the Congress on Evolutionary Computation*, edited by Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala. Mayflower Hotel, Washington D.C., USA: IEEE Press, 1999, volume 1, 788–795.

Ewert, W., W.A. Dembski, and R.J. Marks. "Evolutionary synthesis of nand logic: Dissecting a digital organism." In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*. 2009, 3047 –3053.

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

Golic, J.D. "Techniques for random masking in hardware." *IEEE Transactions on Circuits and Systems I: Regular Papers* 54, 2: (2007) 291–300.

Hanibuchi, T., M. Ueda, K. Higashitani, M. Hatanaka, K. Mashiko, and A. Tada. "A bipolar-PMOS merged basic cell for 0.8 μm BiCMOS sea of gates." *IEEE Journal of Solid-State Circuits* 26, 3: (1991) 427–431.

Healy, Michael. Private communication, 2001.

Hein, James. *Discrete Structures, Logic, and Computability*. Boston: Jones and Bartlett Publishers, 2010.

Ho, Y.-C., and D. L. Pepyne. "Simple Explanation of the No Free Lunch Theorem of Optimization." *Cybernetics and Sys. Anal.* 38, 2: (2002) 292–298.

Jai, Jenq-Neng Hwang, Jai J. Choi, Seho Oh, and Robert J. Marks II. "Query Learning Based on Boundary Search and Gradient Computation of Trained Multilayer Perceptrons*." In *IJCNN 90*. IEEE Press, 1990, 57–62.

———. "Query-based learning applied to partially trained multilayer perceptrons." *IEEE Trans Neural Netw* 2, 1: (1991) 131–6. http://www.biomedsearch.com/nih/Query-based-learning-applied-to/18276359.html.

Jensen, C.A., M.A. El-Sharkawi, and R.J.I.I. Marks. "Power system security boundary enhancement using evolutionary-based query learning." *Engineering Intelligent Systems* 7, 4: (1999) 215–218.

Kennedy, James. *Swarm Intelligence*. San Francisco: Morgan Kaufmann Publishers, 2001.

Lenski, Richard E., Charles Ofria, Robert T. Pennock, and Christoph Adami. "The evolutionary origin of complex features." *Nature* 423, 6936: (2003) 139–144. http://dx.doi.org/10.1038/nature01568.

de Lima, Manoel E., and David J. Kinniment. "Sea-of-gates architecture." *Microelectronics Journal* 26, 5: (1995) 431 – 440. http://www.sciencedirect.com/science/article/B6V44-3YF4G33-27/2/ba003de12f3214847a062b9be0eb2974.

Mackay, David. *Information Theory, Inference, and Learning Algorithms*. Cambridge: Cambridge University Press, 2004.

Marks II, Robert. *Handbook of Fourier Analysis & Its Applications*. Oxford Oxfordshire: Oxford University Press, 2009.

Montañez, George, Winston Ewert, William Dembski, and Robert Marks. "A Vivisection of the ev Computer Organism: Identifying Sources of Active Information." *BIO-Complexity* 2010, 0. http://bio-complexity.org/ojs/index.php/main/article/view/BIO-C.2010.3.

Oh, Seho, Robert J. Marks Ii, and M. A. El-Sharkawi. "Query Based Learning in a Multilayered Perceptron in the Presence of Data Jitter." In *Proc. First International Forum on Applications of Neural Networks to Power Systems*. IEEE, 1991, 72–75.

Papoulis, Athanasios. *Probability, Random Variables, and Stochastic Processes*. New York: McGraw-Hill, 1991.

Phanendra, B., et al. "Clustering with evolution strategies." *Pattern recognition* 27, 2: (1994) 321–329.

Reed, Russell, and Robert J. Marks II. *Neural Smithing*. Cambridge: The MIT Press, 1999.

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

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

Shannon, C. E. "A mathematical theory of communication." *Bell system technical journal* 27.

Spears, William M. (Moderator). "Yin-yang: No-free-lunch theorems for search." In *International Conference on Genetic Algorithms (ICGA-95)*. 1995. http://www.aicnrlnavy.mil.

Stewart, William. *Probability, Markov Chains, Queues, and Simulation*. Princeton: Princeton University Press, 2009.

Strachan, I.G.D. "An Evaluation of Ev." *International Society for Complexity, Information, and Design* http://www.iscid.org/papers/Strachan_EvEvaluation_062803.pdf.

Takita, K., and Y. Kakazu. "Automatic agent design based on gate growth-application to wallfollowing problem." In *SICE'98. Proceedings of the 37th SICE Annual Conference. International Session Papers*. 1998, 863–868.

———. "Evolutionary design of autonomous agent based on gate growth." In *1999 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1999. IROS'99. Proceedings*. 1999, 1555–1560.

Tomasz, Dominik. *Genetic Algorithms Reference*. City: Tomasz Gwiazda, 2006.

Wolpert, David H., and William G. Macready. "No free lunch theorems for optimization." *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION* 1, 1: (1997) 67–82.

Yasunaga, M., T. Nakamura, and I. Yoshihara. "Sonar spectrum recognition chip designed by evolutionary algorithm." In *International Joint Conference on Neural Networks(IJCNN'99)*. 1999, 3182–3187.