42nd South Eastern Symposium on System Theory
University of Texas at Tyler
Tyler, TX, USA, March 7-9, 2010

Efficient Per Query Information Extraction from a Hamming Oracle

Winston Ewert
& George Montañez
Department of Computer Science
Baylor University
Waco, TX

William A. Dembski
Southwestern Baptist
Theological Seminary
Fort Worth
Texas

Robert J. Marks II
Department of Electrical
and Computer Engineering
Baylor University
Waco, TX

***Abstract*—Computer search often uses an *oracle* to determine the value of a proposed problem solution. Information is extracted from the oracle using repeated queries. Crafting a search algorithm to most efficiently extract this information is the job of the programmer. In many instances this is done using the programmer's experience and knowledge of the problem being solved. For the Hamming oracle, we have the ability to assess the performance of various search algorithms using the currency of query count. Of the search procedures considered, blind search performs the worst. We show that evolutionary algorithms, although better than blind search, are a relatively inefficient method of information extraction. An algorithm methodically establishing and tracking the frequency of occurrence of alphabet characters performs even better. We also show that a *search for the search* for an optimal tree search, as suggested by our previous work, becomes computationally intensive.**

I. Introduction

An oracle [6], [11], [12], [13], [21] is a common source of information in assisted search. The computational overhead of the oracle often dominates the time required for a search. In such cases, the computational overhead of a search can be measured by its query count.

Branch-and-bound search [2] for feature selection [18] uses the structure of combinatoric search spaces with the specific goal of reduction of the query count. Using data generated by the *EPRI Electric Transient Mid-Term Stability Program* software [9], Jensen *et al.*, for example, use branch-and-bound search for feature selection to train neural networks. Each query in the search is computationally expensive because it requires the training of a layered perceptron [22].

Simulation software is often used as an oracle. NASA's evolutionary design software of an X-band antenna [16], [17], for example, requires evaluation of fitness during its search process. It does so using *The Numerical Electromagnetics Code* (NEC-4) [3], [4] as an oracle. NEC-4 is a widely used antenna modeling software package for wire and surface antennas. Likewise, ONR developed evolutionary search for optimal ensonification parameters uses repeated queries to an acoustic simulator: a layered perceptron neural network [22] trained on data generated by acoustic modeling software [14].

Oracles can be used with relative degrees of efficiency [10]. 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 a more complex case of assisted search, a "ratchet search" in the software known as Avida results in better per query performance that does a standard evolutionary search [10].

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*

$$\texttt{METHINKS}\star\texttt{IT}\star\texttt{IS}\star\texttt{LIKE}\star\texttt{A}\star\texttt{WEASEL}$$

has $L = 28$ 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 [6], [19], [23]. 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 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) [8] 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.

II. Information Measures

Let the probability of identifying the correct phrase with a single random query be p. The inherent difficulty of the search then can be measured by the *endogenous information* [6],

$$I_\Omega = -\log_2 p.$$

For L letters from an alphabet of size N, we have

$$p = N^{-L}$$

and

$$I_\Omega = L \log_2 N. \tag{1}$$

The endogenous information can be viewed as the total information available from the search.

We consider searches that are *asymptotically perfect*, that is, will eventually succeed. For a given search, let the expected number of queries before a success be Q. Then the *active information per query* [6], [10] is[1]

$$I_\oplus = \frac{I_\Omega}{Q}. \tag{2}$$

Using (1), (2) becomes

$$I_\oplus = \frac{L \log_2 N}{Q}. \tag{3}$$

In a random search with replacement, for example, the number of queries prior to a success is a geometric random variable with mean

$$Q = \frac{1}{p} = N^L.$$

Using (3),

$$I_\oplus = \frac{L \log_2 N}{N^L}.$$

Without replacement, $Q = \frac{1}{2}N^L$ and the active information per query doubles.

III. Markov Models of Evolutionary Search

Markov models are useful in describing certain evolutionary searches [1], [24], [25]. Before analysis, a brief background on Markov processes is required.

A. *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

$$(\mathbf{P})_{k,\lambda} = p_{k,\lambda} \tag{4}$$

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

$$\vec{\pi}(n) = \mathbf{P}^n \vec{\pi}(0) = \left[\begin{array}{c|c} \mathbf{S}^n & \vec{0} \\ \hline \vec{q}_n^{\,T} & 1 \end{array}\right] \vec{\pi}(0). \tag{5}$$

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

$$\sum_{n=0}^{\infty} \mathbf{S}^n = [\mathbf{I} - \mathbf{S}]^{-1}. \tag{6}$$

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

$$(\mathbf{F})_{k,\lambda} = \begin{cases} \text{expected number of the times} \\ \text{state } \lambda \text{ has gone to state } k. \end{cases} \tag{7}$$

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

[1] We are consistent in the use of this definition. Dividing by the expected value of queries, however, does not give the expected value of $I_\oplus$. Indeed, from Jensen's inequality [5], $E[1/Q] \geq 1/E[Q]$.

[2] In some cases, the matrix $\mathbf{I}-\mathbf{S}$ may be ill-conditioned [20]. In our analysis, we only consider inversion of matrices with condition numbers not exceeding 10^5.

chosen initialization. Let G denote the expected number of generations. Then

$$G = \vec{N}^T \vec{\pi}(0) \tag{8}$$

B. Information Measures in Markov Searches

The endogenous information of a search for L letters using an N character alphabet is given in (1). 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 (8). So the active information per query is

$$I_\oplus = \frac{L \log_2 N}{KG}. \tag{9}$$

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 (8) to calculate the expected number of iterations of the algorithm and thus determine the active information per query extracted through the algorithm.

C. Markov Search

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

(A) **K Children Each With a Single Mutation.** 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.

- 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.

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

    $$\Pr[W] = \Pr[W|g]\Pr[g] + \Pr[W|b]\Pr[b]$$
    $$= \frac{N-1}{N} \times \frac{\lambda}{L} + 0 \times \frac{L-\lambda}{L} = \frac{(N-1)\lambda}{NL}. \tag{10}$$

    b) **Better.** The probability of a better score is

    $$\Pr[B] = \Pr[B|g]\Pr[g] + \Pr[B|b]\Pr[b]$$
    $$= 0 \times \frac{\lambda}{L} + \frac{1}{N} \times \frac{L-\lambda}{L} = \frac{L-\lambda}{NL}. \tag{11}$$

    c) **Same.** The probability of the same score is

    $$\begin{aligned} \Pr[S] &= \Pr[S|g]\Pr[g] + \Pr[S|b]\Pr[b] \\ &= \frac{1}{N} \times \frac{\lambda}{L} + \frac{N-1}{N} \times \frac{L-\lambda}{L} \\ &= \frac{\lambda + (N-1)(L-\lambda)}{NL}. \end{aligned} \tag{12}$$

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

    a) **Worse.** The only way to get a worse score is for all the children to have a worse score. Thus, using (10),

    $$\Pr[\lambda - 1|\lambda] = (\Pr[W])^K = \left(\frac{(N-1)\lambda}{NL}\right)^K. \tag{13}$$

    b) **Better.** The probability of at least one of the children having a better score than the parent

[3]As $K \to \infty$, we expect to always have a better result. This is confirmed by the asymptotic values of probabilities in (13), (14) and (15). a) **Worse.** From (13), $\lim_{K\to\infty} \Pr[\lambda - 1|\lambda] = 0$. b) **Better.** From (14), $\lim_{K\to\infty} \Pr[\lambda + 1|\lambda] = 1 - \lim_{K\to\infty} [[1 - (L-\lambda)/(NL)]^K = 1$. c) **Same.** Lastly, from (15), $\lim_{K\to\infty} \Pr[\lambda|\lambda] = \lim_{K\to\infty} (1 - (L-\lambda)/(NL))^K - \lim_{K\to\infty} [((N-1)\lambda)/(NL)]^K = 0$

is the complement of none of the children having a better score. Thus, using (11),

$$\Pr[\lambda+1|\lambda] = 1 - \left(\Pr[\bar{B}]\right)^K$$
$$= 1 - (1-\Pr[B])^K = 1 - \left(1 - \frac{L-\lambda}{NL}\right)^K \tag{14}$$

c) **Same.** Since

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

we conclude from (13) and (14) that

$$\Pr[\lambda|\lambda] = \left(1 - \frac{L-\lambda}{NL}\right)^K - \left(\frac{(N-1)\lambda}{NL}\right)^K. \tag{15}$$

With the state λ as a source node, this analysis can be interpreted using the Markov chain shown on the left in Figure 1 with transitional probabilities given by (13), (14) and (15). As shown in the middle of Figure 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 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.

a) **Coming from state $\lambda + 1$.** Using (13), we obtain

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

b) **Coming from state $\lambda - 1$.** Likewise, with $\lambda \to \lambda - 1$, (14) gives

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

c) **Staying at state λ.** With (15), all transitions from the λ node in the center entry in Figure 1 are now identified.

- **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(A) shows a plot of the active information per query where the choice of K is optimized at each point.

(B) **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.

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

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

b) **Better.** The probability of a better score is the same as before in (11)

$$\Pr[B] = \frac{L-\lambda}{NL}. \tag{16}$$

c) **Same.** The probability of the same score can be derived by adding (12) and (10), since the score will remain the same if the mutation is either deleterious or neutral.

$$\begin{aligned}\Pr[S] &= \frac{(N-1)\lambda}{NL} + \frac{\lambda + (N-1)(L-\lambda)}{NL} \\ &= \frac{\lambda + (N-1)L}{NL}.\end{aligned} \tag{17}$$

- **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.

a) **Coming from state $\lambda - 1$.** With $\lambda \to \lambda - 1$, (16) gives

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

b) **Staying at state λ.** From (17),

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

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

- **Results.** Figure 2(B) shows the active information per query for the ratchet strategy given different alphabet sizes and message lengths. Increasing the message length does not appear to significantly change the efficiency of active information extraction. However, increasing the alphabet size has a rather noticeable effect. The efficiency of information extraction decreases. The plot, however, shows better results than those in Figure 2(A).

(C) **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.

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

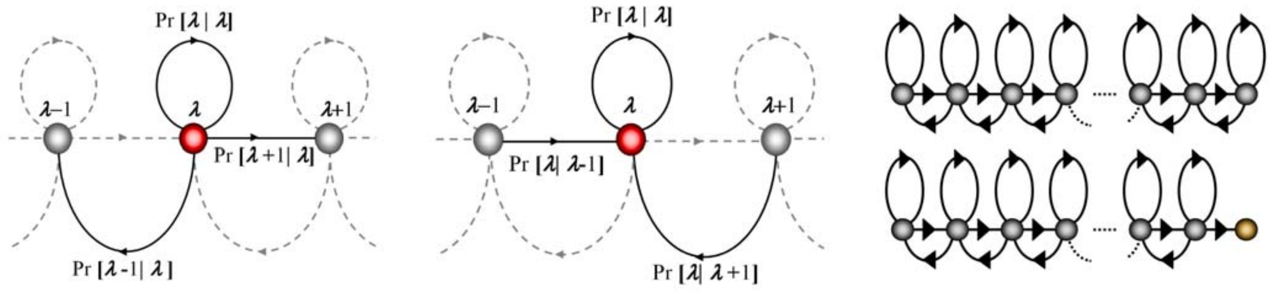


Fig. 1. LEFT: The search problem as a Markov chain. The λ node is shown as a source node. CENTER: An equivalent representation of the Markov model with λ as a sink node. RIGHT: Non absorbing (top) and absorbing (bottom) cases.

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

$$\begin{aligned} p_w &:= \Pr[W|g] \\ &= \Pr[(W|g)|M]\Pr[M] + \Pr[(W|g)|\bar{M}]\Pr[\bar{M}] \\ &= \frac{N-1}{N}\mu + 0 = \frac{\mu(N-1)}{N} \qquad (18) \end{aligned}$$

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 $0 \le \omega \le \lambda$,

$$\begin{aligned} \Pr[k_w = \omega] &= \binom{\lambda}{\omega} p_w^{\omega} (1-p_w)^{\lambda-\omega} \\ &= \binom{\lambda}{\omega} \left(\frac{\mu(N-1)}{N}\right)^{\omega} \\ &\quad \times \left(1 - \frac{\mu(N-1)}{N}\right)^{\lambda-\omega} \qquad (19) \end{aligned}$$

b) Define the probability of *better from bad* as

$$\begin{aligned} p_b &:= \Pr[B|\bar{g}] \\ &= \Pr[(B|\bar{g}|M]\Pr[M] + \Pr[B|\bar{g}|\bar{M}]\Pr[\bar{M}] \\ &= \frac{1}{N}\mu + 0 = \frac{\mu}{N}. \qquad (20) \end{aligned}$$

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 $0 \le \beta \le L-\lambda$,

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

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 $-\lambda \le \kappa \le L-\lambda$, can be written

$$\begin{aligned} f_{\Delta\lambda}(\kappa) &= \sum_{\beta=\max(0,\kappa)}^{\min(\kappa+\lambda, L-\lambda)} \binom{L-\lambda}{\beta} (\mu\pi)^{\beta} \\ &\quad \times (1-\mu\pi)^{(L-\lambda)-\beta} \\ &\quad \times \binom{\lambda}{\beta-\kappa} (\mu(1-\pi))^{\beta-\kappa} \\ &\quad \times (1-\mu(1-\pi))^{\lambda-(\beta-\kappa)} \qquad (21) \end{aligned}$$

where $\pi = 1/N$.
From (21) the cumulative distribution of $\Delta\lambda$, is

$$\begin{aligned} &F_{\Delta\lambda}(x) = \Pr[\Delta\lambda \le x] \\ &= \begin{cases} 0 & ; \quad x < -\lambda \\ \sum_{\kappa=-\lambda}^{x} f_{\Delta\lambda}(\kappa) & ; \quad -\lambda \le \kappa \le L-\lambda \\ 1 & ; \quad x > L-\lambda. \end{cases} \end{aligned} \qquad (22)$$

- **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

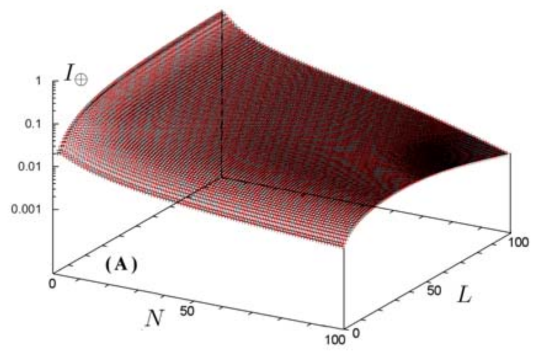
$$F_{\Delta\lambda_{\max}}(x) = \Pr[\Delta\lambda_{\max} \le x] = [F_{\Delta\lambda}(x)]^K \quad (23)$$

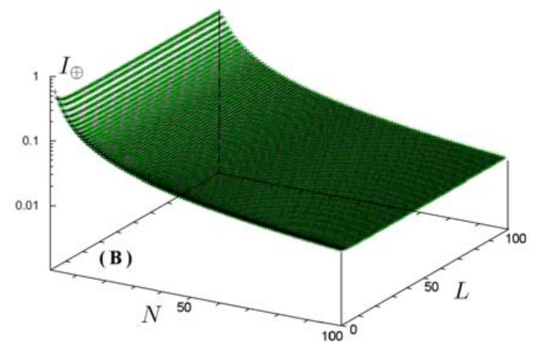
The corresponding probability mass function is

$$f_{\Delta\lambda_{\max}}(x) = F_{\Delta\lambda_{\max}}(x) - F_{\Delta\lambda_{\max}}(x-1). \quad (24)$$

- **Markov Matrix.** The probability mass function in (24) can be explicitly parameterized as $f_{\Delta\lambda_{\max}}(x, \lambda)$. For the mutation example, the Markov matrix then has elements

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





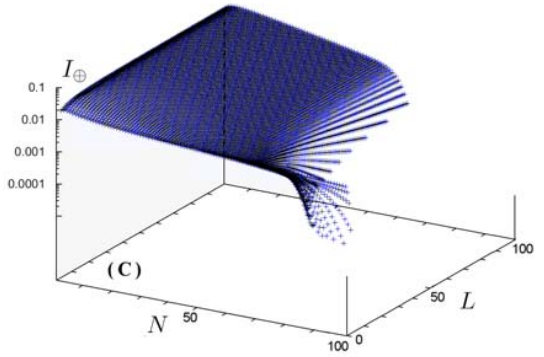
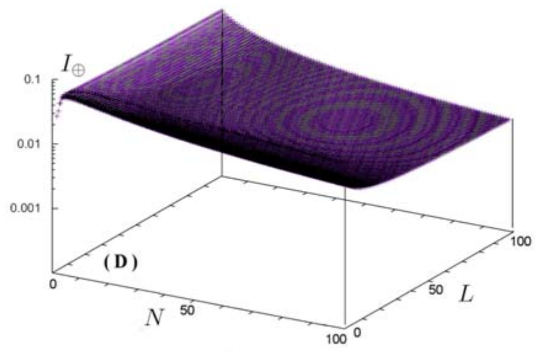


Fig. 2. Active information per query, $I_\oplus$ (in bits), for three evolutionary strategies (A) single-mutation presented in Section III-C(A); (B) ratchet evolutionary strategy discussed in Section III-C(B); and (C) fixed mutation rate ($\mu = .05$) in Section III-C(C). Continued in Figure 3.

- **Results.** As illustrated in Figure 2(C), 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.

(D) **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 (23) and (24)



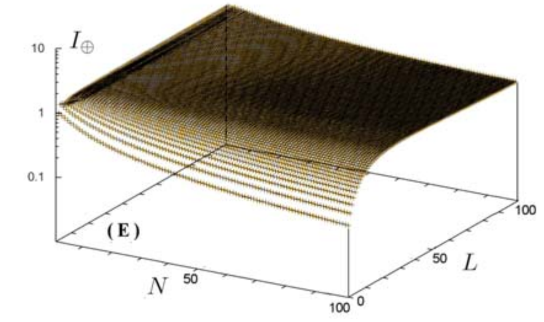


Fig. 3. (D) Active information per query, $I_\oplus$ (in bits), for an evolutionary strategy using optimally scheduled mutation rates presented in Section III-C(D); and (E) Performance of the Frequency Hamming Oracle Algorithm presented in Section IV. Continued from Figure 2.

$$\begin{aligned}
E\left[\Delta\lambda_{\max}\right] &= \sum_{x=-\lambda}^{L-\lambda+1} x f_{\Delta\lambda_{\max}}(x) \\
&= \sum_{x=-\lambda}^{L-\lambda+1} x\left[F_{\Delta\lambda_{\max}}(x) - F_{\Delta\lambda_{\max}}(x-1)\right] \\
&= \sum_{x=-\lambda}^{L-\lambda+1} x\left[F^K_{\Delta\lambda}(x) - F^K_{\Delta\lambda}(x-1)\right] \quad (25)
\end{aligned}$$

where $F_{\Delta\lambda}(x)$ is defined by (21) and (22).

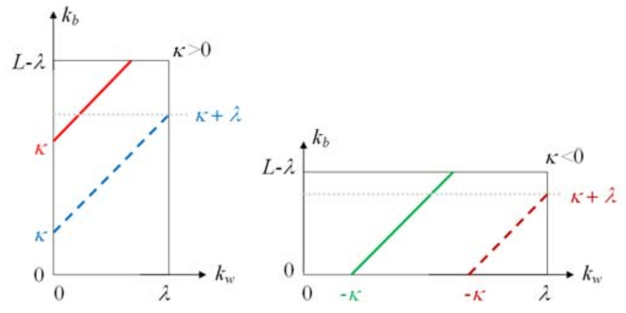


Fig. 4. Determining the limits on the sum in (21). On the left is $\kappa > 0$ and the right is $\kappa < 0$. The lines shown are $k_b - k_w = \kappa$.

- **Results.** For each value of L and N, we have optimized[4] the active information per query as a function of mutation rate, μ for each value of λ. The results are shown in Figure 3(D).

IV. 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 [6] knowledge base. We begin by establishing the FOO of the string. If a string containing all A's is submitted to a 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, $N-1$ 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.
- 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.

[4] An attempt at maximizing (25)) by differentiation with respect to μ and setting to zero results in analytic difficulty. Numerical evaluation of (25) is more straightforward.

[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.

The average active information per query, $I_\oplus$, for the FOO Hamming oracle algorithm is shown in Figure 3(E) as a function of alphabet size, N, and message length, L. The results are significantly better than the Markov results shown in the other plots in Figures 2 and 3. Figure 5 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.

V. 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 [7], [8]. 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.

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.

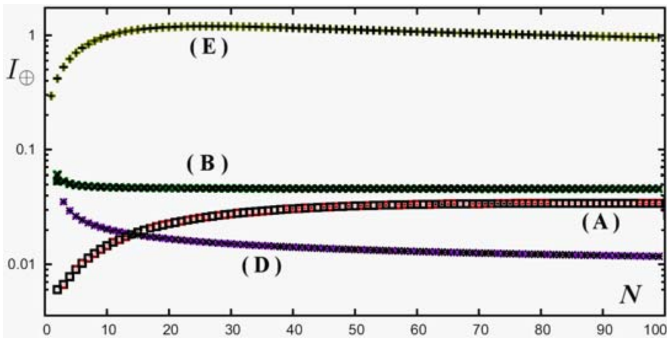


Fig. 5. Averaged active information per query, $I_\oplus$ (in bits), for various algorithms as a function of alphabet size, N. (A) single-mutation presented in Section III-C(A); (B) ratcheted single mutation in Section III-C(B) for one child. (D) Active information per query, $I_\oplus$, for an evolutionary strategy using optimally scheduled mutation rates presented in Section III-C(D) for 100 children; (E) Performance of the FOO Hamming oracle algorithm presented in Section IV operating on a single agent. These plots are obtained from Figures 2 and 3 by averaging over $1 \leq L \leq 100$. The plot in Figure 2(C) is not included because the problem analysis becomes ill-conditioned in places and not all data is available.

$\downarrow L\ N \rightarrow$	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 I
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 II
ACTIVE INFORMATION PER QUERY, $I_{\oplus}$, FROM THE SEARCH ALGORITHM FOUND IN THE S4S OVER ALL SEARCH TREES.

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.

VI. 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/`.

REFERENCES

[1] Thomas Back, **Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms**, Oxford University Press (1996)

[2] Michael Brusco and Stephanie Stahl, **Branch-and-Bound Applications in Combinatorial Data Analysis**, Springer (2005)

[3] Gerald J. Burke, **Numerical Electromagnetics Code NEC-4, Method of Moments, Part I: Users Manual**, Lawrence Livermore National Laboratory, 1992

[4] Gerald J. Burke, **Numerical Electromagnetics Code NEC-4, Method of Moments, Part II: Program Description Theory**, Lawrence Livermore National Laboratory, 1992

[5] T. M. Cover and J. A. Thomas, **Elements of Information Theory**, 2nd ed. New York: Wiley-Interscience, 2006.

[6] William A. Dembski and Robert J. Marks II, "Conservation of Information in Search: Measuring the Cost of Success," IEEE Transactions on Systems, Man and Cybernetics A, Systems & Humans, vol.5, #5, September 2009, pp.1051-1061

[7] William A. Dembski and R.J. Marks II, "Bernoulli's Principle of Insufficient Reason and Conservation of Information in Computer Search," Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics. San Antonio, TX, USA - October 2009, pp. 2647-2652.

[8] William A. Dembski and Robert J. Marks II, "The Search for a Search: Measuring the Information Cost of Higher Level Search," (in review)

[9] Extended Transient Midterm Stability Program, Version 3.0, Palo Alto, California, vol. 16, 1993.

[10] Winston Ewert, William A. Dembski and R.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.

[11] J.N. Hwang, J.J. Choi, S. Oh, R.J. Marks II, "Query learning based on boundary search and gradient computation of trained multilayer perceptrons", Proceedings of the International Joint Conference on Neural Networks, San Diego, June, 1990, 17-21 June 1990, vol. III, pp.III57-III62.©

[12] J.N. Hwang, J.J. Choi, S. Oh and R.J. Marks II, "Query based learning applied to partially trained multilayer perceptrons", IEEE Transactions on Neural Networks, Vol. 2, pp.131-136, (1991).

[13] C.A. Jensen, M.A. El-Sharkawi and R.J. Marks II, "Power Security Boundary Enhancement Using Evolutionary-Based Query Learning", Engineering Intelligent Systems, vol.7, no.9, pp.215-218 (December 1999).

[14] C.A. Jensen, R.D. Reed, R.J. Marks II, M.A. El-Sharkawi, Jae-Byung Jung; R.T. Miyamoto, G.M. Anderson, C.J.Eggen, "Inversion of feedforward neural networks: algorithms and applications," Proceedings of the IEEE, Volume 87, # 9, Sept. 1999, pp. 1536 -1549

[15] C.A. Jensen, M.A. El-Sharkawi and R.J. Marks II, "Power System Security Assessment Using Neural Networks: Feature Selection Using Fisher Discrimination," IEEE Transactions on Energy Conversion, vol.16, no.4, pp.757-763 (November, 2001).

[16] J.D. Lohn, D.S. Linden, G.S. Hornby, A. Rodriguez-Arroyo, S.E. Seufert, B. Blevins, T. Greenling. "Evolutionary design of an X-band antenna for NASA's Space Technology 5 mission," 2004 IEEE Antennas and Propagation Society International Symposium, Volume 3, 20-25 June 2004 pp.2313 - 2316

[17] J.D. Lohn, D.S. Linden, G.S. Hornby, W.F. Kraus. "Evolutionary design of a single-wire circularly-polarized X-band antenna for NASA's Space Technology 5 mission," 2005 IEEE International Symposium Antennas and Propagation Society, Volume 2B, 3-8 July 2005 pp.267 - 270

[18] J. Kittler, A. Etemadi, and N. Choakjarernwanit, "Feature selection and extraction in pattern recognition," in Pattern Recognition and Image Processing in Physics, R. A Vaughan, Ed., 1990, Proceedings of the 37th Scottish University summer school in physics.

[19] D. J. C. MacKay, **Information Theory, Inference and Learning Algorithms**. Cambridge, U.K.: Cambridge Univ. Press, 2002.

[20] R.J. Marks II, **Handbook of Fourier Analysis and Its Applications**, Oxford University Press, (2009).

[21] S.Oh, R.J. Marks II and M.A. El-Sharkawi, "Query based learning in a multilayered perceptron in the presence of data jitter", Applications of Neural Networks to Power Systems, (Proceedings of the First International Forum on Applications of Neural Networks to Power Systems), July 23-26, 1991, Seattle, WA, (IEEE Press, pp.72-75).

[22] Russell D. Reed and R.J. Marks II. **Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks**, (MIT Press, Cambridge, MA, 1999.)

[23] T. D. Schneider, "Evolution of biological information," Nucleic Acids Res., vol. 28, no. 14, pp. 27942799, Jul. 2000.

[24] William M. Spears, **Evolutionary Algorithms: The Role of Mutation and Recombination**, Springer (2004)

[25] William J. Stewart, **Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling**, Princeton University Press (2009)

— ⋉ —

Erratum

The plots in Figure 2 and plot D in Figure 3 were generated incorrectly as the result of at transposition error in the originating code. Discussion of Figure 2(B) in the main paper is incorrect as a result of the error. Increasing either the message length or alphabet size both result in a degradation of performance.

The corrected plots for Figures 2 and 3 in the main paper are shown in Figures 1 and 2 in this Erratum.

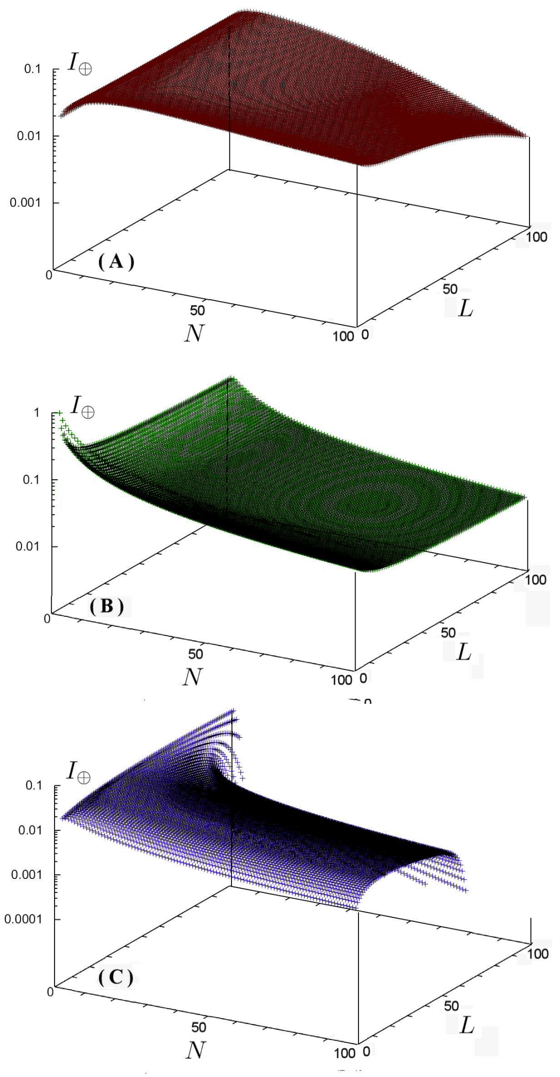


Fig. 1. Erratum for Figure 2 in main paper.

Figure 5 in the main paper was affected by the same transposition as the other plots. A correct version appears in this Erratum in Figure 3.

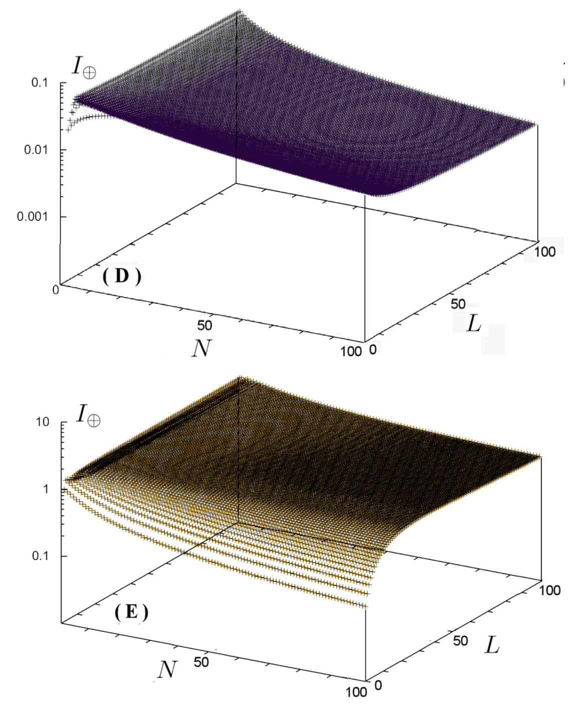


Fig. 2. Erratum for Figure 3 in main paper.

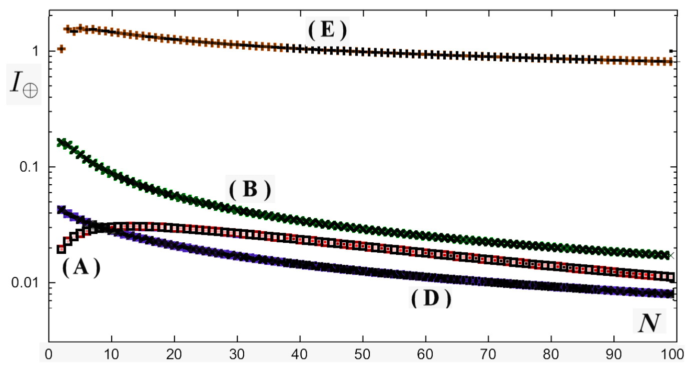


Fig. 3. Erratum for Figure 5 in main paper.

Acknowledgement

The author's thank T. M. English for his interest in our work and for bringing these errors to our attention.