Mastermind, aka "Bulls and Cows," is a codebreaking game. The authors put the game to serious use by testing how well various search algorithms work to solve the game. One result is some guidelines on characteristics of more successful algorithms.

Mastermind as a Test-Bed for Search Algorithms

J.H. O'Geran, H.P. Wynn, and A.A. Zhiglyavsky

We present this piece as an example of research that straddles the two fields in Chance's *subtitle, statistics and computing. While more technical than our usual articles, the material is not too difficult to follow if one reads carefully.*

Search is one of those areas of applied mathematics that impinges on many different areas of science but has not, until recently, had the prominence it deserves. One area of application is computer science, where it forms the basis of many of the inference engines for artificial intelligence. In statistics and probability, it arose first in group testing: finding the sick person by screening out others by pooling blood samples and screening for significant factors using screening designs. The bad-penny problem is perhaps the best-known parlor problem, finding the bad penny out of 12 using weighing scales.

Statistics and probability have much to contribute to this area in the future. There are two main reasons for this. First, entropy and Bayes' methods can help us to understand how to act sequentially, and, second, random strategies are a good, cheap tool for analyzing how well we can search. At its simplest level, we could, for example, compare any strategy with looking at random strategies.

As any computer scientist knows, even simple games can be very hard to analyze if, in some sense, we are looking for intelligent algorithms. The game of Mastermind will provide us with a test-bed to study some algorithms that use statistical principles.

Mastermind (known in some parts of the world as "bulls and cows") is a search game for two players. The first player conceals a "target," which is a code of digits (or colored pegs). The second player then tries to discover the code by sequentially presenting test codes, for each of which he or she receives a score. Each score consists of the number of "bulls" and "cows," a bull being achieved

for each digit that is correct and in the correct position and a cow for each digit that is correct but in the wrong position. The game continues until the second player determines (often with considerable thought) the target code. Here is an example of a simple game in which the target is a four-digit code chosen from the digits 0 to 6. The first line is the target, T, unknown to the second player. Lines 2 to 6 represent the second player's trials X_1 to X_5, and on the right are the bull and cow scores (b,c) achieved by these trials. So, for example, on the third trial, the test code (2354) and the target code (2416) have two digits in common, one in the correct position (the 2) and one in the wrong position (the 4), and hence the score is one bull and one cow:

T	2416	bc
X_1	0123	02
X_2	1045	02
X_3	2354	11
X_4	2406	30
X_5	2416	40

The target is found using five trials (the player used the "first consistent" algorithm, which will be described later).

In general, let the target consist of m digits chosen from a set of d, $T = (t_1, \ldots, t_m)$ say, where $t_i \in \{0, \ldots, d-1\}$ (in the preceding example, $m = 4$, $d = 7$). If $m = 1$, the game is trivial; hence, it is assumed that $2 \le m \le d$. There are two variants of the game. In the "eastern" variant, components within the target T and the test sets $X_1, X_2, \ldots$ must be distinct, whereas in the "western" variant, components may be repeated. We consider the eastern variant.

There are sequential and nonsequential tactics for finding T. In a sequential strategy, the previous k test results may be considered when selecting a new test set X_k for each $k > 1$. In the nonsequential case, the whole series $\{X_1, \ldots, X_N\}$ of test sets is chosen beforehand in such a way that T can be uniquely determined by the test results.

We consider Mastermind a search problem and investigate the properties of algorithms applied to the game. In the next section, Mastermind is defined formally in terms of search notation, along with some examples of similar search problems. In the third section, nonsequential algorithms are discussed and numerically constructed optimal nonsequential designs are given. In the fourth section, sequential algorithms for solving the Mastermind search problem are considered. Some optimality criteria are discussed, and some good sequential algorithms are described with numerical results.

Mastermind as a Search Problem

The general (discrete) search problem can be represented as a triple $(\mathcal{T},\mathcal{X},f)$, where $\mathcal{T} = \{T\}$ is the target field, that is, the set of all possible target combinations T, $\mathcal{X} = \{X\}$ is the test field, that is the set of all possible test combinations, and $f: \mathcal{X} \times \mathcal{T} \to \mathcal{Y}$ is a search function mapping $\mathcal{X} \times \mathcal{T}$ to some space $\mathcal{Y}$. Values $f(X,T)$ for $X \in \mathcal{X}, T \in \mathcal{T}$ are regarded as test or experimental results at a test point X when the unknown target is T. The game of Mastermind has the pleasant property that the target and search fields are the same,

$$\mathcal{T} = \mathcal{X} = \{X = (x_1, \ldots, x_m),$$

$$x_i \in \{0,1, \ldots, d-1\},\ x_i \ne x_j \text{ for } i \ne j\}$$

but the complication that $f(X,T)$ is two-valued,

$$f(X,T) = \begin{pmatrix} b \\ c \end{pmatrix}$$

where $b = b(X,T) = (\#i : x_i = t_i)$ and $c = c(X,T) = (\#i : x_i = t_j, i \ne j)$.

We call b the number of *bulls* and c the number of *cows* and define $a = b + c = (\#i : x_i \in \{t_1, \ldots, t_m\})$ to be the number of *animals*. Thus, the number of animals is the number of correct digits regardless of their position.

In the western variant of the game, where elements of T and X may coincide,

$$\mathcal{T} = \mathcal{X} = \{X = (x_1, \ldots, x_m),$$

$$x_i \in \{0, 1, \ldots, d-1\}\}$$

f, a, and b are determined as previously, and $c = a - b$. This variant of the game is somewhat harder to study. In particular, unlike the eastern variant, the test function $f(\cdot,\cdot)$ is not symmetric; that is, $f(X,T) \ne f(T,X)$ in general. Another way to think of this symmetry is as arising from considering X or T as permutations.

It is instructive to mention some similar search problems. The symbols a, b, and c will be used throughout and are as defined previously.

Case 1: Binary Screening (Group Testing, Bad Penny)

Binary screening or group testing occurs when groups of units are tested simultaneously to discover whether there is at least one defective in the group. Savings in tests are achieved for negative test results because the whole group is then classified as nondefective using a single test. Algorithms are required to screen out all of the defective items in a small number of tests. For example, blood samples may be pooled and tested for the presence of some disease. A positive test signifies that at least one of the patients in the group is infected. We can express this using the "animal" notation a. Let $X = (x_1, \ldots, x_m)$ be the group of units to be tested and $T = (t_1, \ldots, t_s)$ be the set of defectives. Then

$$f(X,T) = \begin{cases} 1 & \text{if } a > 0 \\ 0 & \text{if } a = 0 \end{cases}$$

where $a = (\#i: x_i \in \{t_1, \ldots, t_s\})$ is the number of animals in X.

Case 2: Additive Screening with Equal Effects

The models here are similar to Case 1, differing only in the expression of the test function f, which in this case is additive,

$$f(X,T) = a = (\#i: x_i \in \{t_1, \ldots, t_s\}),$$

that is, the *number* of defectives (or significant factors) in the test set. This corresponds, for example, to an additive model in which all factors giving a nonzero effect have an equal, known effect. Obviously, the test information in Case 2 is more detailed than in Case 1.

The following two models are variants of Mastermind itself.

Case 3: Bulls Only

Define the game as previously, except that only the number of bulls is given by the test function; that is, $f(X,T) = b$. This distance is essentially the Hamming distance for the multilevel case and is familiar from coding theory.

Case 4: Cow Distances

This is an analogue of the Mastermind game with more detailed information on the response. Let ρ be the distance on $\{0,1, \ldots, d-1\}$ defined by $\rho(i,j) = \min\{|i-j|, d-|i-j|\}$ and define for given l, $0 \le l \le [m/2]$, where $[\cdot]$ denotes the integer part of, $X = (x_1, \ldots, x_m)$, and $T = (t_1, \ldots, t_m)$, the number of cows *at distance* l by $c_l = (\#i,j: t_i = x_j, \rho(i,j) = l)$. The test function f is then given by $f(X,T) = (c_0, \ldots, c_{\lfloor m/2 \rfloor})$.

Different variations of the Mastermind game might be considered as variants of the binary screening problem in Case 1 with some partial information on which units are defective. We can see from this collection of problems that Mastermind is a close relative of some quite important problems in statistics and has a surprisingly strong link (via the Hamming distance) to information theory. Indeed, we shall see that *entropy*, which derived from statistical mechanics and took root in information theory, is a very useful tool.

Nonsequential Algorithms

In statistics, experimental design helps us select *where* to observe in an investigation. The notation $Y = f(X,T)$ should prompt us to think of X as a design point (treatment combination) and T as simply an unknown parameter. A fixed set of X_i's is then just an experimental design. Thus, a nonsequential algorithm is a collection of test sets $X_1, \ldots, X_N$, *which are chosen before* the game begins [the test results $f(X_1,T), \ldots, f(X_t,T)$ are *not* used in the selection of X_{t+1}]. *The matrix that has as its* $(i,j)^{\text{th}}$ entry the j^{th} component of the test set X_i is called the *design*. The test function is observed for all test sets in the design, and the number of tests in the game is fixed.

To ensure that the target is found, it is required that all $T \in \mathcal{T}$ must be distinguishable by the design; that is for any pair $T_j, T_k \in \mathcal{T}$, $T_j \neq T_k$, there must be at least one i, $1 \le i \le N$, such that $f(X_i,T_j) \neq f(X_i,T_k)$. This is exactly the parameter identifiability idea; we have enough data to estimate, or in this case find, the parameter.

Table 1 presents examples of optimal nonsequential designs in the sense that the tabulated collections $\{X_1, \ldots, X_N\}$ can distinguish any $T \in \mathcal{T}$ and N is minimal over all such collections, although these designs are not unique. The designs were mostly obtained using an entropy-guided global computer search, which we do not describe here, and for the remaining (d,m), analytic construction was possible. These are fixed strategies

Table 1—Optimal Nonsequential Designs for Various Values of m and d

Test set	$m=3$ $d=4$	$m=3$ $d=6$	$m=3$ $d=8$	$m=3$ $d=10$	$m=4$ $d=4$	$m=4$ $d=6$	$m=4$ $d=8$	$m=4$ $d=10$
X_1	012	012	012	012	0123	0123	0123	0123
X_2	310	251	170	470	3021	5210	3402	8542
X_3	213	530	053	527	0132	4025	2076	5917
X_4	—	452	361	983	1302	2351	1265	7381
X_5	—	—	436	158	—	3012	5607	9471
X_6	—	—	247	134	—	4102	0234	1026
X_7	—	—	—	096	—	—	7032	3297
X_8	—	—	—	—	—	—	4652	7405
X_9	—	—	—	—	—	—	—	7630
X_{10}	—	—	—	—	—	—	—	3765

for which any T can be found (of course, given enough time).

Sequential Algorithms

A sequential algorithm is a procedure that generates a sequence of test sets $X_1, X_2, \ldots, X_N$, which are used to determine T. For each $t > 1$, a sequential algorithm considers the previous test results $f(X_1,T)$, $f(X_2,T), \ldots, f(X_t,T)$ when selecting the next test set X_{t+1}. For a given algorithm, the number $N = N(T)$ of observations needed to determine the target T uniquely is called the *search-length* and will generally depend on $T \in \mathcal{T}$. Therefore, an algorithm may therefore be characterized by the complete set of search-lengths given by the target sets in T, $\{N(T)\}_{T \in \mathcal{T}}$, which we call the *search-length frequency distribution.* Two important characteristics of the search-length frequency distribution are the average length (assuming that the target T is chosen at random)

$$E(N) = \frac{1}{|\mathcal{T}|} \sum_{T \in \mathcal{T}} N(T)$$

(where $|\mathcal{T}|$ is the number of target sets $T \in \mathcal{T}$) and the maximal length $N_{\max} = \max\{N(T), T \in \mathcal{T}\}$. Tables 2–7 give, for various values of (m,d), the distribution $\{N(T)\}_{T \in \mathcal{T}}$ and the average length $E(N)$ for various algorithms. These tables also present the relative efficiency $E(N)/E(N^*)$, where $E(N^*)$ is the minimal average length among the lengths of all algorithms investigated.

Many algorithms were applied to this problem. Those we consider most interesting will be described. First, some definitions will be given that expose important ideas that arise in all useful algorithms in this general area of search.

Suppose that observations made at test sets $X_1, \ldots, X_t$ give rise to test results $f(X_1,T) = y_1, \ldots, f(X_t,T) = y_t$. Then the set

Table 2—Search-length Characteristics for $m = 3$, $d = 4$

Algorithm	Search-length frequency distribution 1	2	3	4	$E(N)$	$\frac{E(N)}{E(N^*)}$
1	1	10	11	2	2.58	1.066
2	1	7	16	—	2.63	1.087
3	1	10	11	2	2.58	1.066
4	1	10	13	—	2.50	1.033
5	1	10	13	—	2.50	1.033
6	1	12	11	—	2.42	1.000
7	1	12	11	—	2.42	1.000
8	1	12	11	—	2.42	1.000

Table 3—Search-length Characteristics for $m = 3$, $d = 6$

Algorithm	Search-length frequency distribution 1	2	3	4	5	$E(N)$	$\frac{E(N)}{E(N^*)}$
1	1	9	66	44	—	3.28	1.127
2	1	12	76	29	2	3.15	1.082
3	1	10	78	31	—	3.15	1.082
4	1	14	78	27	—	3.09	1.062
5	1	14	87	18	—	3.02	1.038
6	1	16	93	10	—	2.93	1.007
7	1	10	95	14	—	3.02	1.038
8	1	16	95	8	—	2.91	1.000

$$\mathcal{T}_t^* = \{T' \in \mathcal{T}: f(X_1,T') = y_1, \ldots, f(X_t,T') = y_t\}$$

is called the *consistent set* of target sets with respect to the previous data, and any member of $\mathcal{T}_t^*$ is called *consistent.* The idea here is that each observation $f(X_j,T), j = 1, 2, \ldots, t$, can be thought of as putting constraints on the possible candidates for the target, namely, that only those $T' \in \mathcal{T}$ that are consistent with the previous data remain candidates for T.

The set $\mathcal{T}_t^*$ characterizes the uncertainty concerning the location of the target at the t^{th} step of an algorithm (in particular, if $|\mathcal{T}_t^*| = 1$, then the target is found). The aim at each step of an algorithm is to reduce the uncertainty. For each $X_{t+1} \in \mathcal{X}$, we can predict the reduction in uncertainty by considering the partition

$$\mathcal{T}_t^* = \bigcup_{y \in \mathcal{Y}} \{T_y' \in \mathcal{T}_t^*: f(X_{t+1},T_y') = y\} \quad (1)$$

where $\mathcal{Y}$ is the set of possible outcomes of $f(X,T)$; that is, $\mathcal{T}_t^*$ is par-

Table 4—Search-length Characteristics for $m = 3$, $d = 10$

Algorithm	Search-length frequency distribution 1	2	3	4	5	6	7	8	$E(N)$	$\frac{E(N)}{E(N^*)}$
1	1	11	46	82	341	205	34	—	5.09	1.184
2	1	12	69	176	293	119	46	4	4.82	1.121
3	1	8	62	222	341	84	2	—	4.60	1.070
4	1	9	78	247	362	23	—	—	4.43	1.030
5	1	14	72	206	305	86	36	—	4.67	1.086
6	1	14	79	304	318	4	—	—	4.30	1.000
7	1	12	71	287	349	—	—	—	4.35	1.012
8	1	14	79	306	316	4	—	—	4.30	1.000

Table 5—Search-length Characteristics for $m = 4$, $d = 4$

Algorithm	Search-length frequency distribution 1	2	3	4	5	$E(N)$	$\frac{E(N)}{E(N^*)}$
1	1	1	6	16	—	3.54	1.117
2	1	4	8	9	2	3.29	1.038
3	1	4	5	14	—	3.33	1.050
4	1	4	9	10	—	3.17	1.000
5	1	4	10	7	2	3.21	1.013
6	1	4	9	10	—	3.17	1.000
7	1	4	9	10	—	3.17	1.000
8	1	4	9	10	—	3.17	1.000

titioned according to which target sets would still be consistent after any particular outcome of the test $f(X_{t+1},T)$. Obviously, some of the sets in the partition (1) may be empty. The fineness of the partition (1) determines the reduction in uncertainty at the $(t+1)^{th}$ step. Thus, a test set X_{t+1} which generates the finer partition should be considered preferable. As an example, let $m = 3$, $d = 5$, and suppose that the first observation is

T	???	b,c
X_1	0 1 2	2,0

Then the consistent set $\mathcal{T}_1^*$ is

$T_1 = (0,1,3)$ $T_2 = (0,1,4)$ $T_3 = (0,1,5)$
$T_4 = (0,3,2)$ $T_5 = (0,4,2)$ $T_6 = (0,5,2)$
$T_7 = (3,1,2)$ $T_8 = (4,1,2)$ $T_9 = (5,1,2)$

(the subscripts on T are unimportant). Now consider taking $X_2 = T_1 = (0,1,3)$. Then $T_1, \ldots, T_9$ are partitioned into groups according to the value of $f(X_2,T_i)$ as follows:

b, c	Group	n_i = size of group
3 0	T_1	1
2 0	T_2, T_3	2
1 0	T_5, T_6, T_8, T_9	4
1 1	T_4, T_7	2

One criterion for measuring the fineness of a partition is entropy. Suppose that the current consistent set $\mathcal{T}_t^*$ contains n elements and that a test set X_{t+1} partitions $\mathcal{T}_t^*$ into l subsets of size $n_1, \ldots, n_l$, where $n_i > 0, 1 \le i \le l$ and $n_1 + \ldots + n_l = n$ (illustrated in the preceding example). Then the *entropy* of the partition is defined as

$$\text{ent}(\{n_1, \ldots, n_l\}) = \log n - \sum_{i=1}^{l} \frac{n_i}{n} \log n_i \qquad (2)$$

The one-step entropy criterion is that the larger the entropy of a partition caused by a test set X_{t+1}, the better the test set.

A second criterion is pair-splitting. If $\mathcal{T}_t^*$ contains n elements, then the total number of pairs of target sets (T_i,T_j), $i \neq j$ in $\mathcal{T}_t^*$ is $n(n-1)/2$. Suppose that X_{t+1} partitions $\mathcal{T}_t^*$ into subsets of sizes $n_1, \ldots, n_l$. Then the number of pairs split by X_{t+1} (that is, distinguished by being sent to different subsets of the partition) is

$$\frac{n(n-1)}{2} - \sum_{i=1}^{l} \frac{n_i(n_i-1)}{2} \qquad (3)$$

By the pair-splitting criterion, the larger the value of (3), the better the test set X_{t+1}.

Several algorithms for solving the Mastermind problem will be

described roughly in order of simplicity of implementation. For tabulated values of m and d, no computation is required to obtain the test sets for Algorithm 1 (although the stopping rule is not necessarily trivial), whereas Algorithm 8 requires a large computer search to obtain the test set at each stage.

Algorithm 1. Sequential nonsequential

Table 1 gives nonsequential designs for solving the problem for various m and d. These are collections of test sets that can always find T. It will not always be necessary, however, to test all of the sets to find T. This is because for some targets $T \in \mathcal{T}$, the outcome of the test function at some of the test sets will be *predictable* from the other test results. For example, if $X_1 = T$, then a score of m bulls will be obtained on the first test, thus uniquely determining T. Therefore, one might sequentially test sets from a nonsequential design, stopping when enough information has been obtained to uniquely determine T. Such an algorithm we call sequential nonsequential. Algorithm 1 selects X_t from the relevant design in Table 1 as the test set at the t^{th} step.

Algorithm 2. First consistent

A common strategy for playing Mastermind seems to be to choose as the next test set a combination that, given the previous test results, is a possible candidate for T, that is, a consistent set. Algorithm 2 chooses at the $(t+1)^{th}$ step ($t = 0,1, \ldots$) the first member of $\mathcal{T}_t^*$ as the next test set X_{t+1} (where it is supposed that the members of $\mathcal{T}_t^*$ are listed in lexicographical order, for example, with $m = 4$, $d = 10$, this is 0123, 0124, . . . , 6789).

Algorithm 3. Entropy-best sequential nonsequential

As with Algorithm 1, this algorithm selects test sets from the nonsequential designs given in Table 1. In this case, however, at the $(t+1)^{\text{th}}$ step the test set from the relevant design is chosen that produces the finest partition (by the entropy criterion) of the consistent set $\mathcal{T}_t^*$.

Algorithm 4. Entropy-best sequential nonsequential plus first consistent

This algorithm is a combination of Algorithms 2 and 3, choosing as the next test set either the first consistent set or a set from the relevant nonsequential designs in Table 1, to give the maximal entropy partition on $\mathcal{T}_t^*$.

Table 6—Search-length Characteristics for $m = 4$, $d = 6$

	Search-length frequency distribution							
Algorithm	1	2	3	4	5	6	$E(N)$	$\frac{E(N)}{E(N^*)}$
1	1	11	89	169	68	22	3.99	1.191
2	1	17	144	171	27	—	3.57	1.066
3	1	23	129	160	45	2	3.64	1.087
4	1	24	157	168	10	—	3.45	1.030
5	1	19	174	162	4	—	3.41	1.018
6	1	22	185	152	—	—	3.36	1.003
7	1	16	182	159	2	—	3.40	1.015
8	1	22	187	150	—	—	3.35	1.000

Table 7—Search-length Characteristics for $m = 4$, $d = 10$

	Search-length frequency distribution											
Algorithm	1	2	3	4	5	6	7	8	9	10	$E(N)$	$\frac{E(N)}{E(N^*)}$
1	1	7	86	709	1517	1548	759	331	68	14	5.69	1.262
2	1	15	211	1240	2108	1203	252	10	—	—	5.01	1.111
3	1	9	139	1151	2446	1127	161	6	—	—	5.00	1.109
4	1	11	202	1580	2543	666	37	—	—	—	4.75	1.053
5	1	16	228	1555	2688	542	10	—	—	—	4.70	1.042
6	1	12	259	2086	2500	180	2	—	—	—	4.51	1.000
7	1	11	228	1946	2685	169	—	—	—	—	4.55	1.009
8	1	12	261	2086	2496	184	—	—	—	—	4.51	1.000

Algorithm 5. Entropy-best consistent
The consistent set that produces the partition of $\mathcal{T}_t^*$ with the maximum entropy is chosen as the next test set X_{t+1}.

Algorithm 6. One-step entropy
All sets in $\mathcal{T}$ are compared, and that which produces the partition of $\mathcal{T}_t^*$ with the maximum entropy is chosen as the next test set X_{t+1}. The test sets in $\mathcal{T}$ are searched lexicographically, beginning with the consistent sets $\mathcal{T}_t^*$, and if more than one set gives the optimal entropy partition, then the first is taken as $X_t + 1$.

Algorithm 7. One-step pair-splitting
This is as in Algorithm 6, except that the pair-splitting criterion (3) is taken as the optimality criterion instead of entropy.

Algorithm 8. One-plus-two-step entropy
As in Algorithm 6, all sets in $\mathcal{T}$ are compared in terms of the entropy of the partition of $\mathcal{T}_t^*$ they induce. If a single set obtains the maximal entropy, then it is taken as the next test set X_{t+1}. If, however, more than one set is optimal, then a second step is considered in the following way. Let $\mathcal{X}_t^*$ be the collection of test sets that give the optimal entropy partition on $\mathcal{T}_t^*$, and let $\mathcal{T}_t^*(X)$ be the partition of $\mathcal{T}_t^*$ given by X. Then for each $X^* \in \mathcal{X}_t^*$, compute the sum of the entropy of the optimal partitions of the subsets in $\mathcal{T}_t^*(X^*)$. That $X^* \in \mathcal{X}_t^*$ for which this sum is maximal is taken to be X_{t+1}.

All algorithms finish at time $t = N$ such that $\mathcal{T}_N^*$ contains exactly one member, this being the target. Tables 2–7 compare the characteristics of Algorithms 1–8 for $d = 4, 6, 10$, $m = 3, 4$.

Discussion

Some broad conclusions may be drawn from these results that can be tested on similar models, leading, we hope, to some useful guidelines for building algorithms in similar settings.

As expected, the sequential algorithms perform better than the fixed nonsequential algorithms. A substantial gain is obtained in expected (not maximum) search-length by using a fixed algorithm in a sequential manner. Entropy is a very valuable tool in search, and this is borne out in this case. We were pleased to find that an alternative criterion such as pair-splitting worked almost as well and sometimes better. This is because both criteria hinge on the fineness of the partition produced by an observation. At first sight, it may seem paradoxical that Algorithm 5 (entropy and consistency) performs worse than Algorithm 6 (pure entropy). On closer inspection, it is clear that an inconsistent set can sometimes serve as a better test set than a consistent set: A good predicted "value" in $\mathcal{T}$ is not necessarily a good test "value" in $\mathcal{X}$. Consistency is a very basic idea, however, and very cheap for large problems. Even for small problems, the gain in expected length seems remarkable. One point is that, at least for smaller problems, increasing the window size to allow two-step-ahead entropy gives very little improvement. The one-step-ahead entropy algorithm seems close to the optimum solution.

There is much work to be done applying these methods to more complex problems and to study matters in computer science such as memory usage, speed of computation, and more complicated heuristic learning strategies.

Additional Reading

O'Geran, J.H., Wynn, H.P., and Zhigljavsky, A.A. (1991), "Search," *Acta Applicande Mathematicae*, 25, 241–276.