Efficient Linkage Discovery by Limited Probing

Robert B. Heckendorn
heckendo@uidaho.edu
Department of Computer Science, University of Idaho, Moscow, ID
83844-1010, USA
Alden H. Wright
wright@cs.umt.edu
Department of Computer Science, University of Montana, Missoula, MT
59812, USA

Abstract

This paper addresses the problem of discovering the structure of a fitness function from binary strings to the reals under the assumption of bounded epistasis. Two loci (string positions) are epistatically linked if the effect of changing the allele (value) at one locus depends on the allele at the other locus. Similarly, a group of loci are epistatically linked if the effect of changing the allele at one locus depends on the alleles at all other loci of the group. Under the assumption that the size of such groups of loci are bounded, and assuming that the function is given only as a "black box function", this paper presents and analyzes a randomized algorithm that finds the complete epistatic structure of the function in the form of the Walsh coefficients of the function.

1 Introduction

Function optimization algorithms can be viewed as a search through a domain space of the function for a value that yields a maximum value in the range space of the function. In a computer, the search is dictated by the representation of the domain and the search operators on that representation. In this paper, we assume a domain of fixed-length binary strings. In this domain, search often proceeds by modifying the bits of previously evaluated points in the search space. Understanding how the bits in the representation interact with each other in defining the value of the function is critical to understanding the function to be optimized. This interaction is called epistatic linkage or epistasis [1]

This paper addresses the problem of determining the epistatic linkage of a function from binary strings to the reals. There is a close relationship between the Walsh coefficients of the function and "probes" (or perturbations) of the function. This relationship leads to two linkage detection algorithms that generalize earlier algorithms of the same type. A rigorous complexity analysis is given of the first algorithm. The second algorithm not only detects the epistatic linkage, but also computes all of the Walsh coefficients. This algorithm is much more efficient than previous algorithms for the same purpose.

[1] For this paper we do not draw a distinction between epistasis and linkage.

2 Background

In very simple fitness functions each bit in the domain independently contributes to the total value of the function. In optimizing these simple fitness functions, each bit can be tested independently against a fixed background of other bits to determine the contribution of that bit. Proceeding through all the bits the optimum can be found in linear time with respect to the number of bits.

Most practical functions are not nearly as simple. For many, the contribution of a bit in the domain to the value of the function is non-linear in that it is dependent on the state of one or more other bits in the domain. This **linkage** effect is called **epistasis** and can be succinctly defined:

> "...if the effect of one unit is not predictable unless the value of another unit is known, then the effects are epistatic... in other words, *the effect of a unit is context dependent*" (Brodie, 2000).

Applied to the case of evolutionary computation, the "units" in the quote above refers to the positions in the problem representation whose values are selected from an **alphabet**. The more units, or positions, that simultaneously interact (the higher the epistasis) the greater the degree of freedom to "hide" the optimum anywhere in the subdomain formed by the interacting units (Heckendorn and Whitley, 1999). Sets of units that epistatically interact are called **epistatic blocks**. For example consider the function f defined over three bits $b_2b_1b_0$ defined as $f(b_2b_1b_0) = b_2 * b_1 + b_0$. The value of b_2's contribution to the function is unaffected by the value of b_0 however, b_2's contribution is dependent on the value of b_1. Therefore, $\{b_2, b_1\}$ forms one epistatic block and b_0 forms a second epistatic block. These blocks are **separable** because they do not share any bits and form subproblems which can be solved separately much like the previous independent bit example. Now consider function $g(b_2b_1b_0) = b_2 * b_1 + b_1 * b_0$. This time we have two epistatic blocks each of two bit positions, but the blocks overlap. Even with this overlap, the value of b_2's contribution to the function is unaffected by the value of b_0 and so there is no epistasis between b_0 and b_2. Overlapping blocks form an overlying constraint satisfaction problem, but not a fundamental problem of epistasis. The two aspects of a problem: epistasis and pattern of overlapping epistatic blocks, define the **structure** of a problem.

A large number of bits of epistasis is no guarantee of a difficult problem. Nor is low epistasis a guarantee of an easy problem. In fact, 3-MAXSAT problems are examples of problems of low epistasis in which all epistatic interactions are known and they are provably NP-complete (Papadimitriou, 1994; Heckendorn, 1999). This means that even if one is given the complete epistatic structure of a function for free a problem may be intractable. Still, knowing the location of epistatically interacting blocks of bits may be used to guide a search for the optimum or the formulation of a representation (Munetomo and Goldberg, 1999b; Munetomo and Goldberg, 1999a; Kargupta and Park, 2001).

If the function is separable, each component can be solved separately. If the function is close to separable, this can guide the choice of crossover operators. In this case, Mühlenbein and Mahnig (Mühlenbein and Mahnig, 1999)

also suggest applying the UMDA algorithm where each component makes up a string position with a higher-order alphabet. Mühlenbein, Mahnig, and Rodriguez (Mühlenbein et al., 1999) give a factorized distribution algorithm (FDA) that applies to additively decomposed functions (that we call embedded landscapes in this paper). This is an example of an estimation-of-distribution algorithm, and the information produced by the algorithms of this paper should be very useful in this type of algorithm.

This paper uses the assumption that the order of epistatic interaction between loci is bounded. In the terminology of this paper, the fitness function is assumed to have k-bounded epistasis. This is equivalent to an assumption that the Walsh coefficients of order greater than k of the fitness function are zero. This assumption is satisfied by some important classes of real-world fitness functions and some commonly used classes of test functions. These include the k-deceptive functions of (Goldberg et al., 1993), the NK-fitness landscape (Kauffman, 1993), the k-CNF MAXSAT problems (Hogg et al., 1996; Rana et al., 1998), and constraint-satisfaction problems (Braunstein et al., 2003).

The general problem of discovering epistatic linkage has been addressed directly and indirectly by many papers. Munetomo and Goldberg showed a simple direct perturbational approach to generalized linkage discovery over a binary alphabet in (Munetomo and Goldberg, 1999b) (Munetomo and Goldberg, 1999a). This basic approach has been extended in (Munetomo, 2002b) and (Munetomo, 2002a). These papers also summarize some other approaches to the problem, and further references are given in Section 10.1. Kargupta et al. (Kargupta and Park, 2001) have shown that for epistatically bounded functions, f, where the epistasis is known to be bounded by k bits, all the Walsh coefficients, a direct measure of the magnitude of epistasis, can be computed in time $O(L^k)$, where L is the length of the representation.

In this paper we present a theoretical framework for the detection of epistatic linkage and the computation of Walsh coefficients for epistatically bounded functions. The Walsh coefficients completely describe the function and so completely characterize the epistatic linkage. The algorithms we present in this paper are **black box algorithms** in that they assume minimal prior knowledge of the function being analyzed. This paper deals with perturbation methods, or what we call **probes**. We give a randomized algorithm for linkage detection which is based on our theoretical framework, and we give rigorous complexity bounds for this algorithm. We extend this to another randomized algorithm that both detects linkage and computes the Walsh coefficients. The algorithm is analyzed under the assumption that the subfunctions of the function are of maximum order k, the support for the subfunctions is chosen randomly, and the number of subfunctions grows linearly with the string length.

3 Our Notation

The space of all bit strings of length L is denoted by $\mathcal{B}$. The binary operators on $\mathcal{B}$ include $\wedge$ which denotes bitwise AND, and $\oplus$ which denotes bitwise

EXCLUSIVE-OR. An overbar (e. g. $\overline{m}$) denotes 1's complement. A string of all ones is denoted $\vec{1}$. Since the L-bit binary representations of the integers in the interval $[0, 2^L)$ coincide with the elements of $\mathcal{B}$, a bit string may be denoted by the corresponding integer. For example, the integer 2^k, $0 \leq k < L$ corresponds to the bit string with a single one in position k, where bit positions are labeled from the right starting at 0. Thus, $2^2 \equiv 0000100$ for $L = 7$. It is convenient to think of a bit string i as corresponding to the set of bit positions indicated by the `1` bits in i. Thus, we write $i \subseteq j$ (i **is contained in** j) when the set corresponding to i is contained in the set corresponding to j, i. e., when $i \wedge j = i$. If $i \subseteq j$ and $i \neq j$ we write $i \subset j$. The **unitation** or **bit count** function $bc(i)$ of string i is the number of ones in i. Given a mask $m \in \mathcal{B}$, let the set $\mathcal{B}_m = \{i \in \mathcal{B} : i \subseteq m\}$. Note $|\mathcal{B}_m| = 2^{bc(m)}$. Brackets are used to denote an **indicator function**: if $expr$ is an expression that may be true or false, then

$$[expr] = \begin{cases} 1 & \text{if } expr \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

4 Walsh Analysis and Embedded Landscapes

Walsh analysis provides a powerful way of looking at the interaction between bits (Heckendorn and Whitley, 1999). In this section we introduce some of the major ideas in Walsh Analysis that we will be using.

Any function $f : \mathcal{B} \rightarrow \mathbb{R}$ can be written as a linear combination of Walsh functions:

$$f(x) = \sum_{i \in \mathcal{B}} w_i \psi_i(x)$$

where the i^{th} Walsh function is defined as:

$$\psi_i(x) = (-1)^{bc(i \wedge x)}$$

and the w_i are referred to as **Walsh coefficients**. The **Walsh transform** is a linear transform of the Walsh coefficients represented as a vector w in $\mathbb{R}^{2^L}$ to the function space f in $\mathbb{R}^{2^L}$. This is a change of basis transformation corresponding to the matrix Ψ with $\Psi_{i,j} = \psi_i(j)$.

$$f = \Psi w \qquad \text{and} \qquad w = \frac{1}{2^L} \Psi f \tag{1}$$

It is not hard to show that Ψ is symmetric and $\Psi\Psi = 2^L I$ where I is the identity matrix.

f **depends on** a bit position k, $0 \leq k < L$, if there exists a $j \in \mathcal{B}$ such that $f(j) \neq f(j \oplus 2^k)$. In other words, f depends on bit position k if flipping bit k changes the value assigned to some string j. The **support** of f is the set of loci that f depends on. The **support mask** of f is a bitstring in $\mathcal{B}$ with `1` bits in exactly and only those positions that support f. By the definition the support mask of ψ_i is i.

An **embedded landscape** is a function $f : \mathcal{B} \rightarrow \mathbb{R}$ which can be written in the form $f = \sum g_j$ where each subfunction g_j has a support mask m_j. Normally, there will be some restriction on the support set masks m_j. The function $f : \mathcal{B} \rightarrow \mathbb{R}$ has k**-bounded epistasis** if it can be written as the sum of subfunctions each of whose support is a set of at most k bits. It has been shown, perhaps most recently in (Heckendorn, 2002):

Theorem 1 *(K-bounded Landscape Theorem) A function $f : \mathcal{B} \rightarrow \mathbb{R}$ has k-bounded epistasis if and only if $w_j = 0 \; \forall \; bc(j) > k$*

Thus, f has k-bounded epistasis if and only if all of its Walsh coefficients of order greater than k are zero. The function f is **linear** if it has 1-bounded epistasis. The function f is **additively separable** if it can be written as a sum of at least two subfunctions where the supports of all subfunctions are pairwise disjoint.

5 Probes

A probe is a way of determining epistatic properties of a function $f : \mathcal{B} \rightarrow \mathbb{R}$ by performing a series of specific function evaluations. For example, in order to determine if the first and third bits of the domain of a 16 bit function are epistatically interacting, the function can be evaluated at these four points:

$$f(1x1xxxxxxxxxxxx)$$
$$f(1x0xxxxxxxxxxxx)$$
$$f(0x1xxxxxxxxxxxx)$$
$$f(0x0xxxxxxxxxxxx)$$

The x's represent a constant **background** bit pattern that does not vary from evaluation to evaluation. If the difference between the first two functions evaluations is different from the difference between the second two function evaluations then we know the bits are interacting. This process of probing function values can be formalized and generalized in the concept of a probe.

More specifically, a **probe** is:

$$P(f, m, c) = \frac{1}{2^{bc(m)}} \sum_{i \in \mathcal{B}_m} (-1)^{bc(i)} f(i \oplus c)$$

where $m \in \mathcal{B}$ and $c \in \mathcal{B}_{\overline{m}}$. m is a bit mask that identifies the bits to be tested for epistatic interaction and c is the static **background** of bits for the probe. The **order of the probe** is number of ones in the mask, or $bc(m)$. The direct computation of the value of a probe requires $2^{bc(m)}$ function evaluations. The constant $\frac{1}{2^{bc(m)}}$ may be ignored for some purposes, but is required if Walsh coefficients are to be calculated.

Theorem 2 *(Walsh Function Probing)*
For any $j, m \in \mathcal{B}$ and $c \in \mathcal{B}_{\overline{m}}$,

$$P(\psi_j, m, c) = \begin{cases} \psi_j(c) & \text{if } m \subseteq j \\ 0 & \text{otherwise} \end{cases}$$

Proof:

$$\begin{aligned} P(\psi_j, m, c) &= \frac{1}{2^{bc(m)}} \sum_{i \in \mathcal{B}_m} (-1)^{bc(i)} \psi_j(i \oplus c) \\ &= \frac{1}{2^{bc(m)}} \sum_{i \in \mathcal{B}_m} \psi_{\vec{1}}(i) \psi_j(i \oplus c) \\ &= \frac{1}{2^{bc(m)}} \sum_{i \in \mathcal{B}_m} \psi_{\vec{1}}(i) \psi_j(i) \psi_j(c) \\ &= \frac{1}{2^{bc(m)}} \psi_j(c) \sum_{i \in \mathcal{B}_m} \psi_{\overline{j}}(i) \end{aligned}$$

By the Balanced Sum Theorem for Hyperplanes (Heckendorn and Whitley, 1999) the sum is $2^{bc(m)}$ if $\overline{j} \subseteq \overline{m}$ which is the same as $m \subseteq j$ and is 0 otherwise.

□

A probe is really probing for nonzero Walsh coefficients by adding and subtracting over a set of Walsh coefficients. If the result is nonzero then one of the component Walsh coefficients is nonzero. If it is zero then we can say very little without further information. The following theorem identifies the set of Walsh coefficients.

Theorem 3 *(Probe Subset)*
For any $m \in \mathcal{B}$ and $c \in \mathcal{B}_{\overline{m}}$,

$$P(f, m, c) = \sum_{j \in \mathcal{B}} [m \subseteq j] w_j \psi_j(c)$$

where w_j is the j^{th} Walsh coefficient of f.

Proof:

$$
\begin{aligned}
f &= \sum_{j \in \mathcal{B}} w_j \psi_j \\
P(f, m, c) &= P(\sum_{j \in \mathcal{B}} w_j \psi_j, m, c) \\
&= \sum_{j \in \mathcal{B}} w_j P(\psi_j, m, c) \qquad \text{since } P(f, m, c) \text{ is a linear transformation of } f \\
&= \sum_{j \in \mathcal{B}} [m \subseteq j] w_j \psi_j(c) \qquad \text{by the Walsh Function Probing Theorem}
\end{aligned}
$$

□

A **maximal nonzero Walsh coefficient** is a Walsh coefficient w_m such that $w_m \neq 0$ and $w_j = 0 \; \forall \; j \supset m$.

Corollary 4 *(Maximal Probe)*
If w_m is a maximal nonzero Walsh coefficient, then for any $c \in \mathcal{B}_{\overline{m}}$,

$$P(f, m, c) = w_m$$

Proof: It follows from Theorem 3 that

$$P(f, m, c) = w_m \psi_m(c)$$

And from the definition of a Walsh function: $\psi_m(c) = (-1)^{bc(m \wedge c)} = (-1)^0 = 1$.
□

A probe can be written as a sum of lower-order probes.

Theorem 5 *(Probe Recursion)*
For any function $f : \mathcal{B} \to \mathbb{R}$, any masks $m, n \in \mathcal{B}$ with $n \subseteq m$, and any $c \in \mathcal{B}_{\overline{m}}$:

$$P(f, m, c) = \frac{1}{2^{bc(n)}} \sum_{i \in \mathcal{B}_n} (-1)^{bc(i)} P(f, m \oplus n, i \oplus c)$$

Proof: Any $j \in \mathcal{B}_m$ can be written uniquely as $j = i \oplus u$ where $i \in \mathcal{B}_n$ and $u \in \mathcal{B}_{m \oplus n}$. Thus:

$$
\begin{aligned}
P(f, m, c) &= \frac{1}{2^{bc(m)}} \sum_{j \in \mathcal{B}_m} (-1)^{bc(j)} f(j \oplus c) \\
&= \frac{1}{2^{bc(n)}} \sum_{i \in \mathcal{B}_n} (-1)^{bc(i)} \frac{1}{2^{bc(m \oplus n)}} \sum_{u \in \mathcal{B}_{m \oplus n}} (-1)^{bc(u)} f(u \oplus i \oplus c) \\
&= \frac{1}{2^{bc(n)}} \sum_{i \in \mathcal{B}_n} (-1)^{bc(i)} P(f, m \oplus n, i \oplus c)
\end{aligned}
$$

□

Theorem 6 *(Nonzero Probe Existence) Given a maximal nonzero Walsh coefficient w_m, for any a: $a \subseteq m$ and any $c : c \in \mathcal{B}_{\overline{m}}$, there exists an $i \in \mathcal{B}_{m \oplus a}$ such that*

$$P(f, a, i \oplus c) \neq 0 \qquad \forall c \in \mathcal{B}_{\overline{m}}$$

Proof: By the Maximal Probe Corollary, $P(f, m, c) = w_m \neq 0$ for any $c \in \mathcal{B}_{\overline{m}}$. By the Probe Recursion Theorem applied with $n = m \oplus a$,

$$P(f, m, c) = \frac{1}{2^{bc(n)}} \sum_{i \in \mathcal{B}_n} (-1)^{bc(i)} P(f, m \oplus n, i \oplus c)$$

Thus, there must exist an $i \in \mathcal{B}_n$ such that $P(f, m \oplus n, i \oplus c) = P(f, a, i \oplus c) \neq 0$. □

6 The Linkage Hypergraph

A hypergraph. is a convenient way to think of the interaction between bits. A **hypergraph** is a collection of vertices V together with a family of nonempty subsets E of V called **hyperedges**. The vertex of the hypergraph can be used to represent a set of epistatically dependent bits. A **linkage hypergraph** is a hypergraph that represents all the sets of epistatically linked bits. A set of vertices, each corresponding to mask $m \neq 0$, is a hyperedge if there is a $c \in \mathcal{B}_{\overline{m}}$ such that $P(f, m, c) \neq 0$. Therefore a hyperedge can be identified by the corresponding mask. The **order of a hyperedge** is the number of ones in the mask. Gao proposed a similar graph corresponding to the order-2 hyperedges the interaction graph (Gao, 2003).

In view of Theorem 6, the mask m is a hyperedge if and only if there is a $j \supseteq m$ such that $w_j \neq 0$. Thus, we have the following corollary.

Corollary 7 *If m is a hyperedge of the hypergraph, and if $a \subseteq m$, then a is also a hyperedge.*

The Order-j Linkage Detection Algorithm in Figure 1 constructs the set of order-j hyperedges of the linkage hypergraph. The order-2 version of this algorithm is similar to the LINC algorithm of (Munetomo and Goldberg, 1999b). However, they start with a population of strings. Then each probe is done using one of the strings of the population to provide the background for the probe. In Figure 1 we use a random background. Later in this article we will compare this approach with using a population.

For an arbitrary function f it is impossible to conclude anything conclusively from evaluating f at a subset of points. For example, if f would be k-epistatically bounded except for the function value at one point, then the above algorithm for $j > k$ will return 0 for any probe unless the probe happens to sample the one exceptional point. For a large string length, the probability that this one exceptional point is sampled can be very small.

```
DETECT-LINKAGE(j,N)
begin
    V ← {0, 1, ..., L − 1}
    E ← ∅
    for each mask m with bc(m) = j do
        if m ∉ E then
            for i ← 1 to N do
                c ← a random string in B_m̄
                if P(f, m, c) ≠ 0 then
                    E ← E ∪ {m}
                    break
                end if
            end for
        end if
    end for
    return E
end DETECT-LINKAGE
```

Figure 1: The Order-j Linkage Detection Algorithm using a random background string.

Thus, assumptions on f are needed in order to use the Order-j Linkage Detection Algorithm to make conclusions. The natural assumption is that f is k-epistatically bounded. The following theorems give a worst-case complexity analysis of the Order-j Linkage Detection Algorithm in this case. This will give us an upperbound on the amount of work to guarantee that all order-j hyperedges are detected with at least probability δ.

Theorem 8 *(Nonzero Probe Probability) Let f be k-epistatically bounded and let m be a mask corresponding to an order-j hyperedge of the linkage hypergraph of f. If c is a randomly chosen string in $\mathcal{B}_{\overline{m}}$, then the probability that $P(f, m, c) \neq 0$ is at least 2^{j-k}.*

Proof: Since m is a hyperedge by the Nonzero Probe Existence Theorem there is a u such that $m \subseteq u$ and $w_u \neq 0$. Without loss of generality we can assume that u has the property that $u \subset v \Rightarrow w_v = 0$. By assumption, $bc(u) \leq k$. Theorem 6 shows that there is at least one $i \in \mathcal{B}_{u \oplus m}$ such that $P(f, m, i \oplus b) \neq 0$ for any $b \in \mathcal{B}_{\overline{u}}$. The probability that the randomly selected background c matches some such i on the positions masked by $u \oplus m$ is at least $2^{-bc(u \oplus m)} = 2^{bc(m)-bc(u)} \geq 2^{j-k}$.

□

The lower bound of Theorem 8 cannot be improved for functions that are k-epistatically bounded. To see this, start with a $(j-1)$-epistatically bounded function whose support is m with $bc(m) = k$, and then perturb the value of

one point. Any probe that does not include the perturbed point will return a value of zero. Since an order-j probe includes 2^j points, and since there are 2^k probes, the probability of including the perturbed point is 2^{j-k}.

Theorem 9 *Let f be k-epistatically bounded and let J be the number of order-j hyperedges in the linkage hypergraph of f. If the number of iterations N in the Order-j Linkage Detection Algorithm is chosen so that either*

$$N \geq \begin{cases} \frac{\ln(1-\delta^{1/J})}{\ln(1-2^{j-k})} & \text{if } j < k \\ 1 & \text{if } j = k \end{cases} \tag{2}$$

or

$$N \geq \begin{cases} -2^{k-j}\ln(1-\delta^{1/J}) & \text{if } j < k \\ 1 & \text{if } j = k \end{cases}$$

then the probability that all order-j hyperedges are detected is at least δ.

Proof: In the following, a "success" is the detection of a nonzero probe. Theorem 8 shows that the probability of failure for one probe on one trial is at most $1-2^{j-k}$. Thus, the probability of failure on N independent trials is at most $(1-2^{j-k})^N$, and the probability of success on N trials is at least $1-(1-2^{j-k})^N$. The probability of success on all J hyperedges is at least

$$\left(1-(1-2^{j-k})^N\right)^J$$

Thus, we want to choose N so that

$$\begin{aligned} \left((1-(1-2^{j-k}))^N\right)^J &\geq \delta \\ 1-\delta^{1/J} &\geq (1-2^{j-k})^N \\ \ln(1-\delta^{1/J}) &\geq N\ln(1-2^{j-k}) \\ \frac{\ln(1-\delta^{1/J})}{\ln(1-2^{j-k})} &\leq N \end{aligned}$$

To prove the second formula, we need to show that

$$\begin{aligned} -2^{k-j}\ln(1-\delta^{1/J}) &\geq \frac{\ln(1-\delta^{1/J})}{\ln(1-2^{j-k})} \\ \iff \quad 2^{k-j} &\geq -\frac{1}{\ln(1-2^{j-k})} && \text{by dividing both sides by } -\ln(1-\delta^{1/J}) \\ \iff \quad 2^{j-k} &\leq -\ln(1-2^{j-k}) && \text{by inverting both sides .} \end{aligned}$$

By the Taylor series for $-\ln(1-x)$, we see that $-\ln(1-x) \geq x$, or that $-\ln(1-2^{j-k}) \geq 2^{j-k}$.

These formulas are defined when $j < k$, but fail when $j = k$. Fortunately we know by Maximal Probe Corollary that if the function is indeed k-epistatically bounded a single probe is all that is required.

□

Lemma 10

$$\lim_{J\to\infty} \frac{-\ln(1-\delta^{1/J})}{\ln J} = 1$$

Proof: First, we apply l'Hôpital's rule:

$$\lim_{J\to\infty} \frac{-\ln(1-\delta^{1/J})}{\ln J} = \lim_{J\to\infty} \frac{-\delta^{1/J}\ln\delta}{J(1-\delta^{1/J})}$$

The limit of the numerator is clearly $\ln\delta$.

To evaluate the limit of the denominator, make the variable change $x = 1/J$, and take the limit as $x \to 0$. Then apply l'Hôpital's rule again.

$$\lim_{x\to 0} \frac{1-\delta^x}{x} = \lim_{x\to 0} \frac{\delta^x \ln\delta}{1} = \ln\delta$$

□

We next consider how the number N of iterations increases as the string length increases for a class of fitness functions.

Corollary 11 *Assume a class of k-epistatically bounded fitness functions where the number of maximal Walsh coefficients is $O(L)$. If δ is constant, the number of function evaluations required by the* DETECT-LINKAGE *algorithm is $O\left(2^k\binom{L}{j}\left(\ln L + \ln\binom{k}{j}\right)\right)$. If j is constant, then the number of function evaluations is $O\left(2^k L^j \ln L\right)$.*

Proof: By Theorem 9 it is sufficient to choose N to be $-2^{k-j}\ln(1-\delta^{1/J})$. For each of the N iterations of the inner loop a probe is done that requires 2^j function evaluations. The outer loop is executed $\binom{L}{j}$ times, so the total number of function evaluations is: $-2^k\binom{L}{j}\ln(1-\delta^{1/J})$, and by Lemma 10, this is $O\left(2^k\binom{L}{j}\ln J\right)$.

The number of order-j hyperedges in a single maximal Walsh coefficient of order k is bounded by $\binom{k}{j}$, and we have assumed that the number of maximal (order k) hyperedges is $O(L)$, so J is $O\left(\binom{k}{j}L\right)$. Thus, $\ln J$ is $O\left(\ln L + \ln\binom{k}{j}\right)$.

Note that $\ln\binom{k}{j} \le \ln\left(k^j\right) = j\ln k$. If j is constant, then $j\ln k$ is $O(\ln L)$ so the last result of the corollary follows. □

Strictly speaking, Corollary 11 does not apply when probe backgrounds are chosen from a population (as is the case for the LINC algorithm of (Munetomo and Goldberg, 1999b)) since the above analysis assumes that the backgrounds of probes are chosen independently. However, our empirical results show that these formulas are quite accurate when the backgrounds are chosen from a population. (See Section 11.)

For $j = 2$, this result can be compared to the population sizing result of (Munetomo and Goldberg, 1999b). If r is the probability of successfully detecting a single subfunction, they give the population size needed as approximately $-2^k\ln(1-r)$. This translates into $-2^k L^2\ln(1-r)$ function evaluations.

Since they don't address the question of whether the probabilities of detecting subfunctions are independent of the subfunction, this does not translate into a statement about the overall probability (analogous to our δ) of detecting all subfunctions.

7 Computing the Walsh Coefficients Using the Kargupta-Park Top-down Algorithm

Kargupta and Park (Kargupta and Park, 2001) give a "deterministic" algorithm to find the Walsh coefficients of a function f with k-bounded epistasis. It is "top-down" since it does high-order probes before low-order probes. In this section we show how this algorithm can be expressed in terms of probes.

Let w_m be a maximal nonzero Walsh coefficient. The Maximal Probe Corollary shows that $P(f, m, c) = w_m$ for any $c \in \mathcal{B}_{\overline{m}}$. Thus, if f has k-bounded epistasis, and if we do the probe $P(f, m, 0)$ where $bc(m) = k$, the result will be w_m. Thus, all of the order-k Walsh coefficients can be computed by doing $\binom{L}{k}$ probes, each of which uses 2^k function evaluations.

Let j be a mask with $bc(j) = k - 1$. Then Theorem 3 gives the equation

$$P(f, j, 0) = w_j + \sum_{j \subset u} w_u \psi_u(0) = w_j + \sum_{j \subset u} w_u \tag{3}$$

(Note that $\psi_u(0) = 1$.) The potentially nonzero Walsh coefficients in the summation are all of order k and have been computed. Thus, w_j can be computed from $P(f, j, 0)$ plus these order-k Walsh coefficients. Let m be such that $bc(m) = k$ and $j \subseteq m$. If the Probe Recursion Theorem is applied to $P(f, m, 0)$ with $n = m \oplus j$, then the first term in the summation is $P(f, j, 0)$. This shows that all function evaluations necessary to compute $P(f, j, 0)$ have already been done in the computation of $P(f, m, 0)$. (This observation is ours and is not included in (Kargupta and Park, 2001).)

The same idea can be used to compute the lower-order Walsh coefficients. Thus, the Walsh coefficients are computed in order of decreasing bit count, starting with bit count k.

8 Detecting linkage and computing the Walsh coefficients

Kargupta and Park (Kargupta and Park, 2001) give a "bottom up" randomized algorithm that finds the nonzero Walsh coefficients. They suggest that they can find the values of these nonzero Walsh coefficients, but the method to do this is not included in their algorithm, and so presumably one applies the algorithm referred to in Section 7.

In this section, we give a well-specified algorithm that efficiently finds the nonzero Walsh coefficients and computes their values. The algorithm consists

of two passes. The first proceeds in a bottom-up fashion to find which Walsh coefficients are nonzero, and then it proceeds top-down to determine their values without doing any additional function evaluations. (We assume that function evaluations are disproportionately expensive to compute.)

A key observation is that if probe backgrounds are determined by using a population, as in the Munetomo/Goldberg LINC algorithm, then higher order probes can be computed relatively cheaply by using the function evaluations of previously computed lower order probes. This is justified by Theorem 13 below. In other words, computing $P(f, m, c)$ can be done with only one additional function evaluation as long as the probes for all a, $a \subset m$, have been computed using the same background c.

Lemma 12 *Let $h(a, i)$ be any 2-argument function. Given a value x:*

$$\sum_{a \in \mathcal{B}_x} \sum_{i \in \mathcal{B}_m} h(a, i) = \sum_{i \in \mathcal{B}_x} \sum_{z \in \mathcal{B}_{i \oplus x}} h(i \oplus z, i)$$

Proof:

The two sides of the equation are equal if for a given x they sum over the same *set* of arguments to h. Beginning with the set of argument tuples for the left hand side we transform it into the set of argument tuples for the right hand side.

$$\begin{aligned}
\sum_{a \in \mathcal{B}_x} \sum_{i \in \mathcal{B}_m} h(a, i) &= \sum_{a : a \subseteq x} \sum_{i : i \subseteq a} h(a, i) \\
&= \sum_{a : a \subseteq x} \sum_{i : i \subseteq a \subseteq x} h(a, i) \\
&= \sum_{i : i \subseteq x} \sum_{a : i \subseteq a \subseteq x} h(a, i) \\
&= \sum_{i : i \subseteq x} \sum_{z : i \subseteq i \oplus z \subseteq x, i \wedge z = 0} h(i \oplus z, i) \\
&= \sum_{i : i \subseteq x} \sum_{z : z \subseteq i \oplus x} h(i \oplus z, i) \\
&= \sum_{i \in \mathcal{B}_x} \sum_{z \in \mathcal{B}_{i \oplus x}} h(i \oplus z, i)
\end{aligned}$$

□

Theorem 13 *For any $m \in \mathcal{B}$, $c \in \mathcal{B}_{\overline{m}}$,*

$$f(m \oplus c) = \sum_{a \in \mathcal{B}_m} (-2)^{bc(a)} P(f, a, c)$$

This can be restated as:

$$(-2)^{bc(m)}P(f,m,c) = f(m \oplus c) - \sum_{a \in \mathcal{B}_m \setminus \{m\}} (-2)^{bc(a)}P(f,a,c)$$

Proof:

$$\begin{aligned}
P(f,a,c) &= \frac{1}{2^{bc(a)}} \sum_{i \in \mathcal{B}_a} (-1)^{bc(i)} f(i \oplus c) \\
(-1)^{bc(a)}2^{bc(a)}P(f,a,c) &= (-1)^{bc(a)} \sum_{i \in \mathcal{B}_a} (-1)^{bc(i)} f(i \oplus c) \\
\sum_{a \in \mathcal{B}_m} (-2)^{bc(a)}P(f,a,c) &= \sum_{a \in \mathcal{B}_m} \psi_{\vec{1}}(a) \sum_{i \in \mathcal{B}_a} \psi_{\vec{1}}(i) f(i \oplus c) \\
&= \sum_{a \in \mathcal{B}_m} \sum_{i \in \mathcal{B}_a} \psi_{\vec{1}}(a)\psi_{\vec{1}}(i) f(i \oplus c) \\
\text{Using Lemma 12 with } h(a,i) = \psi_{\vec{1}}(a)\psi_{\vec{1}}(i)f(i \oplus c): \quad &= \sum_{i \in \mathcal{B}_m} \sum_{z \in \mathcal{B}_{m \oplus i}} \psi_{\vec{1}}(z \oplus i)\psi_{\vec{1}}(i) f(i \oplus c) \\
&= \sum_{i \in \mathcal{B}_m} \sum_{z \in \mathcal{B}_{m \oplus i}} \psi_{\vec{1}}(z \oplus i \oplus i) f(i \oplus c) \\
&= \sum_{i \in \mathcal{B}_m} f(i \oplus c) \sum_{z \in \mathcal{B}_{m \oplus i}} \psi_{\vec{1}}(z)
\end{aligned}$$

By the Balanced Sum Theorem for Hyperplanes the inner sum is nonzero only if $\vec{1} \subseteq \overline{m \oplus i}$ which can only happen if $i = m$. Therefore:

$$\sum_{a \in \mathcal{B}_m} (-2)^{bc(a)}P(f,a,c) = f(m \oplus c) \sum_{z \in \mathcal{B}_0} \psi_{\vec{1}}(z) = f(m \oplus c)$$

□

The algorithm takes advantage of previously computed function evaluations by caching all function evaluations in a hash table. When the function f is applied to a bit string, this hash table is checked before doing the actual function evaluation. A second feature of the algorithm is that regions of the bit representation can be shown not to contain any higher order epistasis and hence are not reexamined in later portions of the algorithm.

The basic idea of the bottom-up part of the algorithm (TRAVERSE-HYPERGRAPH) (See Figure 2) is to do a breadth-first traversal of the lattice of masks, starting with the empty mask, then looking at the order-1 masks, order-2 masks, etc. When a new mask m is considered for inclusion in the linkage hypergraph, all

submasks of order $bc(m)-1$ are checked for membership in the hypergraph. If any of these submasks is not in the hypergraph, then m cannot be in the hypergraph. If these tests succeed, then a sequence of probes is done to determine if the mask is in the hypergraph.

Note that the maximum order k of epistasis is not an input to the algorithm. The algorithm determines k. However, the algorithm may not be computationally tractable if there are many high order subfunctions.

The backgrounds of the probes can be determined either by using a population or by randomly choosing background strings. The first element of the population or the first background is the all-zeros string since this simplifies the computation of the Walsh coefficients in the top-down part of the algorithm. If a population is used, the remainder of the population is chosen randomly. The value returned by the probe using the all-zeros background is saved in the hash-table $hypergraph$ which is also used to determine whether a mask has been added to the hypergraph.

In addition to the queue used for the breadth-first traversal, the masks added to the hypergraph are stored in a linked list $hypergraphList$ which is traversed in the top-down part of the algorithm.

TESTBYPROBES(a, N) does up to N probes using the mask a. (This is similar to the Detect-Linkage linkage algorithm of Figure 1 and so is not provided.) If one of these probes is nonzero (or greater than a tolerance in practice), then it returns the probe value corresponding to the all-zeros string. If all probes are zero, then it returns $null$. The value of N can depend on the bit-count of the mask a and can be based on Equation 2 of Theorem 9. Thus, some prior knowledge about k and the number of hyperedges would be useful in order to apply Equation 2. The complexity analysis done in section 8.1 might be useful in this regard.

One way to do this would be to apportion parts of the error probability δ to masks of different order. For example, if $k = 5$, and assuming that one wants the overall probability of error to be less than δ, then one would use $\delta/4$ in Equation 2 for $j = 1, 2, 3, 4$ respectively.

In a practical implementation, one might want to add all masks of up to some cardinality to $hypergraphList$ even if TESTBYPROBES returns $null$. This would be especially true of the empty mask of order 0 since Equation 2 does not apply.

SUPERSET-LIST(m) is a list of masks a such that $bc(a) = bc(m) + 1$ and so that a is obtained by adding a 1 to m to the right of the rightmost 1 of m. Note that if a is a mask in the hypergraph, then it will have a subset m so that $a \in$SUPERSET-LIST(m).

The top-down part of the algorithm (COMPUTE-WALSH-COEFS) (See Figure 3) traverses the hyperedges of hypergraph using the list $hypergraphList$ from higher order masks to lower order, that is in the reverse order from which they were added to the hypergraph. The Walsh coefficients are computed using only the function evaluations done in the bottom-up part of the algorithm.

As an example, suppose that $L = 4$ and the the fitness function is a sum of a function that depends on positions $\{0, 1, 2\}$ and a function that depends on po-

```
TRAVERSE-HYPERGRAPH()
    population.initialize()
    hypergraphList.initialize()
    queue.initialize()
    m ← { }          // Empty mask
    ProbeValue ←TESTBYPROBES(m, N(bc(m)))
    if ProbeValue ≠ null then
        queue.add(m)
        hypergraph[m] ← ProbeValue
    end if
    while queue.notEmpty() do
        m ← queue.remove()
        probeValue ← hypergraph[m]
        for a ∈ SUPERSET-LIST(m) do
            if all subsets of a of cardinality bc(m) are in the hypergraphList then
                ProbeValue ←TESTBYPROBES(a, N(bc(a)))
                if ProbeValue ≠ null then
                    queue.add(a)
                    hypergraph[a] ← ProbeValue
                    hypergraphList.addFirst(a)
                end if
            end if
        end for
    end while
```

Figure 2: The Traverse-Hypergraph algorithm, which is the linkage detection portion of the linkage detection/Walsh coefficient computation algorithm.

sitions $\{1, 2, 3\}$. The masks corresponding to the sets $\{\ \}, \{0\}, \{1\}, \{2\}, \{3\}, \{0, 1\}, \{0, 2\}$ will be added to *hypergraph* and to *hypergraphList*. TESTPROBES will return *null* for the mask corresponding to $\{0, 3\}$. The masks corresponding to $\{1, 2\}, \{1, 3\}, \{2, 3\}, \{0, 1, 2\}$ will be added to *hypergraph* and to *hypergraphList*. The masks corresponding to $\{0, 1, 3\}$ and $\{0, 2, 3\}$ will fail the subset test.

The algorithm is based on Equation 3. This equation would suggest that to compute w_a, one would want to traverse those supersets of a that correspond to hyperedges. However, in the top-down algorithm we are already traversing these superset hyperedges, and it is more efficient to add the Walsh coefficient of each of these superset hyperedges to its subsets, and this is what the algorithm does. In other words, as the supersets of a are traversed in the algorithm, their Walsh coefficients are added to $wCoef[a]$.

COMPUTE-WALSH-COEFS($hypergraphList$)
for $m \in hypergraphList$ **do** //traverse in reverse order from order added
$probeValue \leftarrow hypergraph[m]$
if $wCoef[m] \neq null$ **then** $wCoef[m] \leftarrow wCoef[m] + probeValue$
else $wCoef[m] \leftarrow probeValue$ **end if**
for each $a \subset m$ **do**
if $wCoef[a] \neq null$ **then** $wCoef[m] \leftarrow wCoef[a] - wCoef[m]$
else $wCoef[a] \leftarrow -wCoef[m]$ **end if**
end for
end for

Figure 3: Compute-Walsh-Coefs which is the top-down part of the linkage detection/Walsh coefficient computation algorithm used to calculate the Walsh coefficients.

8.1 Complexity Analysis

In this section we give an analysis of the time-complexity of the TRAVERSE-HYPERGRAPH algorithm in the case where the fitness function is an embedded landscape with randomly chosen components of a fixed order k. We suppose that the number s of such components grows linearly with the string length L.

This covers a large and important practical category of problems but is by no means complete. We assume that k is fixed, and so our analysis is only in terms of the string length L. A random fitness function is chosen by choosing ML order-k masks with replacement from the set of $\binom{L}{k}$ possible masks. All nonzero Walsh coefficients of the fitness function must be contained within these masks. Otherwise, the fitness function is arbitrary within this constraint.

This is almost the class of functions described by (Gao, 2003) as the pure random model $\mathcal{F}(L, s, k)$, except that he assumes that support for subfunctions are chosen randomly without replacement from the $\binom{L}{k}$ possible sets of size k, and we assume random choice with replacement.

Strictly speaking, the analysis of this section applies to the version of the TRAVERSE-HYPERGRAPH algorithm that uses random backgrounds for all probes and that uses no function caching. Under the assumption that the epistasis order k is fixed, this only changes the number of function evaluations by at most the constant factor of 2^k, and thus does not change the asymptotic complexity.

Given a specific fitness function and given a mask a of order r, there are three possibilities for the mask. First, the mask may be a subset of a nonzero Walsh coefficient. We will call this a *type-1 mask*. Second, it may not be a subset of a nonzero Walsh coefficient, but it may have an order $r-1$ submask which is a subset of a nonzero Walsh coefficient. We will call this a *type-2 mask*. Note that TEST-BY-PROBES must always return $null$ when called on a type-2 mask. And third, it may neither be a subset of a nonzero Walsh coefficient nor have

any order $r-1$ submasks which are subsets of nonzero Walsh coefficients. We will call this a *type-3 mask*. Note that a type-3 mask will never be tested in the TRAVERSE-HYPERGRAPH algorithm.

The number of probes done in a call to TEST-BY-PROBES is determined by Equation 2 of Theorem 9, and the number of function evaluations per probe is bounded by 2^r where r is the order of the mask. Corollary 11 shows that the number of probes is $O(\log L)$.

The number of type-1 masks contained in one maximal Walsh coefficient of the fitness function is bounded by 2^k. Thus, under the assumption that k is fixed, the total number of type-1 masks is $O(L)$, and the number of function evaluations done in tests of type-1 masks is $O(L \log L)$.

Given that a mask is of type-2 or type-3, we want to find the probability (over fitness functions satisfying our assumptions) that it is of type-2.

Lemma 14 *Let m be a randomly chosen mask with $bc(m) = k$. Let a be a mask with $bc(a) = r$ such that $a \not\subseteq m$. Let $b \subset a$ where $bc(b) = bc(a) - 1$. The probability that $b \subseteq m$ is given by:*

$$P(b \subseteq m) = \frac{|m : b \subseteq m| - |m : a \subseteq m|}{|m : bc(m) = k| - |m : a \subseteq m|} = \frac{\binom{L-r+1}{k-r+1} - \binom{L-r}{k-r}}{\binom{L}{k} - \binom{L-r}{k-r}}$$

Proof: There are $\binom{L}{k}$ possibilities for mask m with $bc(m) = k$. Of these, $\binom{L-r}{k-r}$ contain a, and $\binom{L-r+1}{k-r+1}$ contain b. Thus, the number of masks m that contain b but do not contain a is $\binom{L-r+1}{k-r+1} - \binom{L-r}{k-r}$. The formula of the lemma follows. □

Lemma 15 *Let m be a randomly chosen mask with $bc(m) = k$. Let a be a mask with $bc(a) = r$ such that $a \not\subseteq m$. Let $b_1, b_2, \ldots, b_w$ be distinct masks with $bc(b_j) = bc(a) - 1$ and $b \subset a$ for all $j = 1, 2, \ldots, w$. Then the probability that $b_j \subseteq m$ for some j is wQ where $Q = \frac{\binom{L-r+1}{k-r+1} - \binom{L-r}{k-r}}{\binom{L}{k} - \binom{L-r}{k-r}}$.*

Proof: Note that $b_j \subseteq m$ implies $b_v \not\subseteq m$ for all $v \neq j$. Lemma 14 shows that $P(b_j \subseteq m) = Q$ for each $j = 1, 2, \ldots, w$, and since these are disjoint events, the probability of their union is the sum of their probabilities. Thus, $P(\exists j \ . \ b_j \subseteq m) = wQ$. □

The Inclusion/Exclusion Principle (Niven, 1965) from combinatorics is provided for reference for the following lemma without proof:

Lemma 16 *Let B_i, $i \in \mathcal{I}$, be a finite collection of sets. Then*

$$P\left(\bigcup_{i \in \mathcal{I}} B_i\right) = \sum_{w=1}^{|\mathcal{I}|} (-1)^{w+1} \sum_{\substack{\mathcal{J} \subseteq \mathcal{I} \\ |\mathcal{J}| = w}} \bigcap_{j \in \mathcal{J}} P(B_j)$$

Note that if $|\mathcal{I}| = 3$, then the above lemma says that

$$P(B_1 \cup B_2 \cup B_3)$$
$$= P(B_1) + P(B_2) + P(B_3) - P(B_1 \cap B_2) - P(B_1 \cap B_3) - P(B_2 \cap B_3) + P(B_1 \cap B_2 \cap B_3)$$

Theorem 17 *Let a be a mask with $bc(a) = r$. Let $m_1, m_2, \ldots, m_s$ be masks with $bc(m_i) = k$ randomly and independently chosen such that $a \not\subseteq m_i$ for any $i = 1, 2, \ldots, s$. (Since the m_i are chosen with replacement, it is possible that $m_i = m_j$ for some $i \neq j$.) The probability that a is a type-2 mask is given by*

$$U(L, k, r, s) = \sum_{j=0}^{r} (-1)^j \binom{r}{j} (1 - jQ)^s \tag{4}$$

where

$$Q = \frac{\binom{L-r+1}{k-r+1} - \binom{L-r}{k-r}}{\binom{L}{k} - \binom{L-r}{k-r}} \tag{5}$$

Proof: Let the order $r-1$ subsets of a be $b_1, b_2, \ldots, b_r$. In other words, $b_1, b_2, \ldots, b_r$ are distinct masks such that $bc(b_j) = bc(a) - 1$ and $b_i \subset a$ for all j.

Let B_j be the event that $b_j \not\subseteq m_i$ for all $i = 1, 2, \ldots, s$. If any B_j happens, then a cannot be a type-2 mask. Thus, if A is the event that a is a type-2 mask, then the complement $\overline{A}$ of A is the union of the B_j. Thus, we have:

$$P(A) = 1 - P\left(\bigcup_{j=1}^{r} B_j\right)$$

We want to apply the formula of Lemma 16, so we want to compute the probability of the intersection of an arbitrary collection of B_js.

Given a subset $\mathcal{J} \subseteq \{1, 2, \ldots, r\}$ with $|\mathcal{J}| = w$, and given a fixed i, Lemma 15 shows that the probability that $b_j \not\subseteq m_i$ for all $j \in \mathcal{J}$ is $1 - wQ$. Since the masks m_i are chosen independently, the probability that for all $i = 1, 2, \ldots, s$, $b_j \not\subseteq m_i$ for all $j \in \mathcal{J}$ is $(1 - wQ)^s$. In other words,

$$P\left(\bigcap_{j \in \mathcal{J}} B_j\right) = (1 - wQ)^s$$

Note that there are $\binom{r}{w}$ cardinality w subsets of $\{1, 2, \ldots, r\}$. Thus, applying the formula of Lemma 16, we get that

$$P(A) = 1 - \sum_{w=1}^{r} (-1)^{w+1} \binom{r}{w} (1 - wQ)^s$$
$$= \sum_{w=0}^{r} (-1)^w \binom{r}{w} (1 - wQ)^s$$

□

A small example: Let $L = 4$, $k = 3$, $s = 3$. In this example, we will write masks as binary strings. Let $a = 0111$ so that $r = 3$. Let $b_1 = 0011$, $b_2 = 0101$, $b_3 = 0110$. Then the m_i are chosen from 3 possible order-3 masks that do not contain a, namely $c_1 = 1011$, $c_2 = 1101$, and $c_3 = 1110$. There are 27 ways to choose the m_i, namely all sequences of three choices from $\{c_1, c_2, c_3\}$ (with replacement). It is not hard to see that a is a type-2 mask if and only if each of c_1, c_2, and c_3 is chosen once. There are 6 out of the 27 ways of choosing the m_is that satisfy this condition, so the probability that a is a type-2 mask is $6/27 = 2/9$.

Applying the formula of Theorem 17 gives

$$Q = \frac{\binom{L-r+1}{k-r+1} - \binom{L-r}{k-r}}{\binom{L}{k} - \binom{L-r}{k-r}} = \frac{\binom{2}{1} - \binom{1}{0}}{\binom{4}{3} - \binom{1}{0}} = \frac{1}{3}$$

and

$P(a$ is a type-2 mask $)$

$$= 1 - 3(1-Q)^3 + 3(1-2Q)^3 - (1-3Q)^3 = 1 - 3 \cdot \left(\frac{2}{3}\right)^3 + 3 \cdot \left(\frac{1}{3}\right)^3 - 0 = \frac{2}{9}$$

Another small example: Let $L = 4$,$k = 2$, $r = 2$, $s = 2$. Without loss of generality, we can choose a particular mask a, and ask what is the probability that it is a type-1 mask, a type-2 mask, or a type-3 mask. Thus, let us choose $a = 0011$. There are 6 possible order-k masks, and thus there are $6^s = 6^2 = 36$ possible fitness functions. Thus, we are interested in the type of mask a for each of these 36 fitness functions. This is given in the following table:

	0011	0101	0110	1001	1010	1100
0011	1	1	1	1	1	1
0101	1	3	2	3	2	3
0110	1	2	3	2	3	3
1001	1	3	2	3	2	3
1010	1	2	3	2	3	3
1100	1	3	3	3	3	3

From the table, we see that the probabilities that a is a type-1, type-2, and type-3 mask are $11/36$, $17/36$, and $8/36$. The probability that a is a type-2 mask given that it is of type-2 or type-3 is $8/25$. This corresponds to Theorem 17 where $Q = 2/5$ and $U(L, k, r, s) = 8/25$.

A larger example: Let $k = 3$ and $s = 4{*}L$. For the sequence $L = \langle 20, 40, 80, 160, 320, 640\rangle$ and $r = 2$, the values of $U(L, k, r, s)$ given by Equation 4 are

$$\langle .999984, .999986, .999987, .999987, .999987, .999988\rangle$$

For the same L sequence and $r = 3$, the probabilities are:

$$\langle .339127, .091055, .017298, .002692, .000376, .000050\rangle$$

Note that there are $\binom{L}{r}$ possible order r probes. Thus, the expected number of order r probes should not exceed $\binom{L}{r}U(L,k,r,s)$, where $U(L,k,r,s)$ is given by Equation 4. For the above L sequence, for $r=2$, these values are

$$\langle 190.00, 779.99, 3159.96, 12719.84, 51039.36, 204477.46\rangle$$

and for $r=3$ these values are

$$\langle 386.61, 899.62, 1421.17, 1803.42, 2036.64, 2165.73\rangle$$

These values are confirmed by experiment.

Lemma 18

$$\sum_{j=0}^{r}(-1)^j\binom{r}{j}j^i = 0 \text{ for } i=0,1,\ldots,r-1$$

$$\sum_{j=0}^{r}(-1)^j\binom{r}{j}j^i = (-1)^r r! \text{ for } i=r$$

Proof: Start with the polynomial

$$(x+1)^r = \sum_{j=0}^{r}\binom{r}{j}x^j \tag{6}$$

Let Φ denote the operator $x\cdot\frac{d}{dx}$. (In other words, the operator Φ takes the derivative with respect to x, and then multiplies by x.) After one application of Φ, we get

$$\Phi((x+1)^r) = xr(x+1)^{r-1} = \sum_{j=0}^{r}\binom{r}{j}jx^j$$

After $i<r$ applications of the operator to the left side of 6, we see that every term includes an $(x+1)$ factor. After $i<r$ applications of the operator to the right side of Equation 6, we get

$$\sum_{j=0}^{r}\binom{r}{j}j^ix^j$$

Substituting $x=-1$ gives the first equation of the lemma.

After r applications of the operator to the left side, we see that there is one term $r!x^r$ that does not include an $(x+1)$ factor. After r applications of the operator to the right side, we get

$$\sum_{j=0}^{r}\binom{r}{j}j^rx^j$$

and again substituting $x=-1$ gives the second equation of the lemma. □

Next we collect some facts that will be useful in the proof of the next theorem.

Lemma 19

$$\binom{L}{r} \leq \frac{L^r}{r!} = O(L^r) \tag{7}$$

$$\binom{ML}{i} \leq \frac{(ML)^i}{i!} = O(L^i) \tag{8}$$

$$Q = \frac{\frac{L-r+1}{k-r+1} - 1}{\prod_{i=0}^{r-1} \frac{L-i}{k-i} - 1} = O(L^{1-r}) \tag{9}$$

$$\left| \sum_{j=0}^{r} (-1)^j \binom{r}{j} j^i \right| \leq \sum_{j=0}^{r} \binom{r}{j} j^i = O(r^i) \tag{10}$$

Proof:

$$\binom{L}{r} = \frac{\prod_{i=0}^{r-1} L - i}{r!} \leq \frac{L^r}{r!}$$

The proof of Equation 8 is similar.

Equation 9 follows by factoring and canceling $\binom{L-r}{k-r}$ from the numerator and denominator of Q.

$$\sum_{j=0}^{r} \binom{r}{j} j^i \leq r^i \sum_{j=0}^{r} \binom{r}{j} = r^i 2^r = O(r^i)$$

□

Theorem 20 *Let k and M be constant.*

For $r = 1$:

$$U(L, k, r, ML) = 1$$

and thus the expected number of order-1 type-2 masks is $O(L)$.

For $r = 2$:

$$\lim_{L \to \infty} U(L, k, r, ML) = 1 - 2e^{-kM} + e^{-2kM}$$

and thus the expected number of order-2 type-2 masks is $O(L^2)$.

For $r = 3$:

$$\lim_{L \to \infty} \binom{L}{r} U(L, k, r, ML) = \frac{1}{6} M^3 k^3 (k-1)^3$$

and thus the expected number of order-3 type-2 masks is $O(1)$.

For $r > 3$:

$$\lim_{L \to \infty} \binom{L}{r} U(L, k, r, ML) = 0$$

and thus the expected number of order-r type-2 masks is $o(1)$.

Proof: For $r = 1$ it is easy to see that $Q = 1$, and substituting into Equation 4 gives $U(L, k, 1, ML) = 1$. There are $\binom{L}{r} = O(L)$ possible order-1 masks, so the expected number of type-2 order-1 masks is $\Theta(L)$.

For $r = 2$, Equation 9 shows that for large L, Q is approximately k/L. Substituting k/L for Q in Equation 4, we have

$$U(L, k, 2, ML) \approx 1 - 2\left(1 - \frac{k}{L}\right)^{ML} + \left(1 - \frac{2k}{L}\right)^{ML}$$

It is well known that $\lim_{n\to\infty}\left(1 - \frac{1}{n}\right)^{nx} = e^{-x}$. A change of variable gives the limit result. Since there are $\binom{L}{r} = O(L^2)$ possible order-2 masks, the expected number of type-2 order-2 masks is $\Theta(L^2)$.

For $r \geq 3$, we have

$$\begin{aligned}
\binom{L}{r}U(L, k, r, ML) &= \binom{L}{r}\sum_{j=0}^{r}(-1)^j\binom{r}{j}(1 - jQ)^{ML} \\
&= \binom{L}{r}\sum_{j=0}^{r}(-1)^j\binom{r}{j}\sum_{i=0}^{ML}\binom{ML}{i}(-1)^i(jQ)^i \quad \text{by the binomial theorem} \\
&= \binom{L}{r}\sum_{i=0}^{ML}\binom{ML}{i}(-1)^iQ^i\sum_{j=0}^{r}(-1)^j\binom{r}{j}j^i \\
&= \sum_{i=r}^{ML}\binom{L}{r}\binom{ML}{i}(-1)^iQ^i\sum_{j=0}^{r}(-1)^j\binom{r}{j}j^i \quad \text{by Lemma 18}
\end{aligned} \tag{11}$$

Given $\epsilon > 0$, we claim the existence of a u such that for any L with $u \leq ML$,

$$\binom{L}{r}\sum_{i=u}^{\infty}\binom{ML}{i}Q^i\sum\binom{r}{j}j^i < \epsilon/2$$

Use Lemma 19, we see that:

$$\begin{aligned}
\binom{L}{r}\sum_{i=u}^{\infty}\binom{ML}{i}Q^i\sum\binom{r}{j}j^i &\leq CL^r\sum_{i=u}^{\infty}L^i\ L^{i(1-r)}r^i \quad \text{for some constant } C \\
&= CL^r\sum_{i=u}^{\infty}\left(rL^{2-r}\right)^i \\
&= CL^r\left(rL^{2-r}\right)^u\sum_{i=0}^{\infty}\left(rL^{2-r}\right)^i \\
&= Cr^uL^{r+u(2-r)}\frac{1}{1 - rL^{2-r}}
\end{aligned}$$

This quantity can be made arbitrarily small by choosing L sufficiently large. Thus, if u is chosen to be sufficiently large, then the above can be made to be less than $\epsilon/2$. This proves the claim.

For $r = 3$, the $i = r = 3$ term of the summation on line 11 is:

$$\frac{L^3}{6}\frac{ML(ML-1)(ML-2)}{6}(-1)\left(\frac{k(k-1)}{L^2}\right)^3(-1)3! + O(L^{-1})$$
$$= \frac{M^3k^3(k-1)^3}{6} + O(L^{-1})$$

We can choose L to be sufficiently large that the $O(L^{-1})$ term above is less than $\frac{\epsilon}{2u}$.

For $r \geq 3$ and $3 < i < u$, the absolute value of the ith term of the summation in Equation 11 is easily seen to be $O(L^{-1})$. Thus, we can choose L to be sufficiently large that this absolute value is less than $\frac{\epsilon}{2u}$.

We have shown that for $r \geq 3$, the summation of line 11 is less than ϵ from the claimed limit. □

Corollary 21 *For randomly chosen fitness functions whose maximal Walsh coefficients are of order k, and where the number of maximal Walsh coefficients is $O(L)$, the expected number of function evaluations of the* TRAVERSE-HYPERGRAPH *and* COMPUTE-WALSH-COEFFICIENTS *algorithms is $O(L^2 \log L)$. (This assumes that k and the algorithm error probability δ are constant as L increases.)*

Thus, under the fitness function model where the maximal Walsh coefficients are at most order k, and where the number of maximal Walsh coefficients grows linearly with L, the TRAVERSE-HYPERGRAPH and COMPUTE-WALSH-COEFFICIENTS algorithms can find the complete structure of fitness function with a probability of error for the entire algorithm of at most δ using $O(L^2 \log L)$ function evaluations.

Note that the hypothesis of linear growth of the number of maximal Walsh coefficients is satisfied by all fitness functions which are sums of nonoverlapping subfunctions of fixed order and by the NK fitness functions when K is fixed. It is also satisfied by k-MAXSAT problems where the ratio of clauses to variables is fixed.

9 Using the Walsh coefficients to define the function

Once the Walsh coefficients are known, the function f is completely known. This section describes how f could be computed in this case.

The basic idea is to represent f as an embedded landscape. This could be done with one subfunction per nonzero Walsh coefficient, but in many cases, it can be done so that evaluation of f is more efficient.

Theorem 22 *The function f can be written as an embedded landscape where there is a subfunction for each maximal Walsh coefficient. The support of a subfunction is the index of the corresponding maximal Walsh coefficient.*

Proof: For each index j such that $w_j \neq 0$, let $a(j)$ be the index of some maximal nonzero Walsh coefficient so that $j \subseteq a(j)$. For each index m of a maximal Walsh coefficient, let

$$g_m = \sum_j [a(j) = m] w_j \psi_j$$

Then clearly $f = \sum_m g_m$.

Let N be the set of maximal nonzero Walsh coefficients, N_i. It is clear that the set of *all* nonzero Walsh coefficients can be partitioned into subsets S_i, one for each N_i such that N_i is in S_i. $\forall w_j \in S_i, w_j \subseteq S_i$. Note that this partitioning is not unique. Each subset S_i defines a subfunction whose support is N_i. The subfunction f_i can be enumerated by performing the inverse Walsh transform on the vector of Walsh coefficients whose indices are contained in the index of N_i.

□

The fast Walsh transform can be used to compute a function table for each subfunction of the embedded landscape. If the subfunction has a support mask of bit count k, then the fast Walsh transform can be computed in time $\Theta(k \log k)$. Then the function f can be computed by summing the values of the subfunctions.

10 How the algorithms of this paper can be applied in practice

The section describes in somewhat imprecise terms how the algorithms of this paper might be used in the more general situation where the assumption of k-bounded epistasis is not true, or is only approximately true.

10.1 Finding the linkage groups

The traditional genetic algorithm motivation for linkage detection algorithms has been the preservation of building blocks (Munetomo and Goldberg, 1999b). Goldberg (Goldberg, 1989) defines a *building block* as a "highly-fit, short-defining-length schema". While the meaning of "highly-fit" is not completely clear, a building block could be defined as a configuration of a small number of loci which is part of a highly fit solution to the problem. Two loci are *tightly linked* if one or more of the Walsh coefficients of some of the masks which contain both loci are relatively large, or equivalently if there is some background so that a probe of the corresponding two-bit mask has relatively large magnitude. A collection of loci is called a *linkage group* if they are pairwise tightly linked.

If a collection of loci form a linkage group, then a configuration of these loci might be a building block. One theory is that genetic algorithms work by assembling building blocks into a high-fitness solution. (This theory is certainly true for some fitness functions. It remains to be seen how widely appli-

cable the theory is.) According to this theory, it is important that the crossover operation not be too disruptive of the building blocks, and this suggests that the crossover operation should be designed so that linkage groups (groups of highly linked loci) are not overly disrupted by the crossover operator.

We suggest the following method to find the linkage groups. Use the order-2 DETECT-LINKAGE algorithm to find the linkage graph, except that we also want to use probes to measure the strength or weight of edges. For each mask of a nonzero probe do a total of N probes. Then some measure of the overall magnitude of these probes will be used to weight the edges of the linkage graph. For example, this could be the maximum absolute value of a probe, or the mean of the absolute values of the probes.

Now consider the sequence of edge weights, sorted in decreasing order. If the fitness function is approximately additively separable, and if there is not "hidden epistasis" which is not detected by the algorithm, then there should be a transition from relatively large edge weights to relatively small edge weights. (Those edges connecting loci which are in different separability components will have relatively small edge weights.) Consider the subgraph of the linkage graph whose edges are the edges with relatively large edge weights. If this graph has multiple components, then the vertices of these components should be the linkage groups. If there is no "natural" transition from relatively large weight edges to small weight edges, then the fitness function is not approximately additively separable. This approach is similar to that proposed by (Munetomo, 2002a).

There has been extensive previous work on the "linkage learning problem". Proposed methods to solve the "linkage learning" problem include perturbational methods of (Munetomo and Goldberg, 1999a) and this paper, the linkage learning genetic algorithm (LLGA) of (Harik and Goldberg, 2000; Chen and Goldberg, 2003), the fast messy GA of Kargupta (Kargupta, 1996), and the ECGA (Extended Compact Genetic Algorithm, an estimation of distribution algorithm) of (Harik, 1999). Other estimation of distribution algorithms such as the MIMIC algorithm of (de Bonet et al., 1997) and the BOA algorithm of (Pelikan et al., 1999) can also solve linkage problems. All of these methods except the perturbational method of (Munetomo and Goldberg, 1999a) address the linkage problem as part of trying to solve an optimization problem and thus have a somewhat different focus from that of this paper and (Munetomo and Goldberg, 1999a). The test functions used in the LLGA and the ECGA approach are predominantly embedded landscapes with nonoverlapping subfunctions where each subfunction is a trap function ((Deb and Goldberg, 1992)). In these papers, sometimes a relatively small number of trap subfunctions are embedded in a long genome with many other nonfunctional (constant) subfunctions, and sometimes the trap functions are nonuniformly scaled. The LLGA approach seems to be limited to learning the linkage for a small (less than 10) number of nontrivial subfunctions if the subfunctions are uniformly scaled.

10.2 An estimation of distribution approach

Another approach is inspired by "estimation of distribution" algorithms (see, for example (Mühlenbein and Mahnig, 1999) or (Pelikan et al., 1999)). The idea is to build a directed acyclic graph model of the fitness function. The linkage graph described above which contains the high-probe-weight edges is the underlying undirected graph. Note that if one does multiple order-1 probes on the loci, then one also can get probe weights on the loci. This gives a natural way to add directions to the edges of the graph—namely the direction of an edge is from the locus of higher probe weight to the locus of lower probe weight.

The idea of an estimation-of-distribution algorithm is to use the graphical model of the fitness function to choose the next generation population. Then selection is applied to this population, and a new graphical model is computed to complete one generational cycle of the algorithm.

If results of the probes for a mask are consistent in sign, then the settings of all but one of the loci for that mask will determine the setting of the remaining locus that will increase the fitness function. For example, if all of the results of an order-1 probe for a 1-bit mask are of one sign, this says that there is one way to set the allele for that locus to increase the value of the fitness function which is consistent over the entire sample used in doing the probes. If all of the results of an order-2 probe over a 2-bit mask are of one sign, this says that the setting of one of the two bits will determine the other bit.

11 Empirical results

In this section, we first give empirical results on the Order-2 Linkage Detection Algorithm of Section 6 to the "linkage learning" problem as it is defined in (Harik and Goldberg, 2000). We give empirical evidence that the formulas of Theorem 9 apply when probe backgrounds are chosen from a population (rather than randomly as in the theorem), and to the algorithm of Section 8. Finally, we give some results that show the size of problems that can be done on commonly available hardware at the time that this paper was written.

The class of subfunctions that requires the largest number N of probes for the algorithms of this paper are "needle-in-the-haystack" functions which are constant except at a single point. For these functions, an order-1 probe will return zero unless one of the two points evaluated in doing the probe is the needle point. MAXSAT and one-max are problems whose subfunctions are of this type.

The class of subfunctions requiring the next largest number of probes are the subfunctions that are linear except at a single point. For these functions, an order-2 probe will return zero unless one of the four points evaluated in doing the probe is the exceptional point. The concatenated trap functions of (Deb and Goldberg, 1992) and the deceptive functions of (Goldberg et al., 1993) can be of this type.

One of test functions used in this section is an embedded landscape where the subfunctions are linear with a randomly placed "needle". The coefficients of the linear function are chosen randomly from the interval $[0, 1]$. The needle is a single point with a value of 0.1 greater than the corresponding value given by the linear function. The support of each subfunction is randomly chosen when the subfunctions are overlapping, and randomly chosen subject to the nonoverlapping constraint when the functions are nonoverlapping. These functions are the same difficulty as trap functions for the algorithms of this paper. (A trap function is a linear function with a systematically placed needle.)

The algorithms of this paper were coded in Java and run on PC hardware. Sean Luke's "Mersenne twister" Java random number generator (`http://www.cs.umd.edu/users/seanl`) was used since the random number generator supplied with Java was found to have dependencies which affected the results.

11.1 The Order-2 algorithm applied to nonoverlapping subfunctions

Our Order-2 Linkage Detection Algorithm, with sufficient CPU time as described by the formulas of Theorem 9, can resolve the epistatic structure of a function with any number of subfunctions[2]. To illustrate this, we ran the Order-2 Linkage Detection algorithm with randomly generated background strings on 800-bit functions with 160 order-5 nonoverlapping subfunctions. Each subfunction was a linear function with a randomly placed needle. The linkage graph of each subfunction is a complete graph on 5 vertices, and so the linkage graph for the whole fitness function consists of 160 components, each of which is a complete graph on 5 vertices.

The first formula of Theorem 9 says that 95 probes per potential linkage graph edge are needed to achieve a 0.99 probability of detecting all edges of the linkage graph, and 112 probes per edge are required to achieve 0.999 overall success probability.

However, the algorithm can solve the linkage learning problem (for nonoverlapping subfunctions) with considerably less probes per potential edge since all edges do not need to be detected to identify the components of the true linkage graph. This can be done by finding enough edges so that the components of the discovered linkage graph are the same as the components of the true linkage graph. This is illustrated by the results given below for the Order-2 Linkage Detection Algorithm. The string length was 800, and a values less than 10^{-10} was considered to be zero, and the 160 subfunctions were identically scaled. A run was considered successful only if all 160 components of the linkage graph were successfully identified.

[2]As stated earlier even if the epistatic structure of a problem is completely determined the problem may, of course, remain NP-complete and hence intractable.

Number of probes per edge	Runs	Successes	Number of function evaluations per run
15	200	164	19,176,000
20	200	197	25,568,000
30	200	200	38,352,000

The CPU time per run for 15 probes per edge was about 35 minutes on a 2 GHz. Pentium© 4, and double that for 30 probes/edge. The amount of memory used was about 36 megabytes in either case (some of this was due to data structures relating to checking whether the results were correct).

11.2 Overlapping subfunctions with backgrounds from a population

In this subsection we give empirical evidence that drawing background strings from a population does not change the complexity results of Section 8.1.

The test function used in this subsection is an embedded landscape with 50 5-bit subfunctions and a string length of 50. Each subfunction is linear with a randomly placed "needle". A value is considered to be zero if it is less than 10^{-7}.

The algorithm used is that given in Section 8. The algorithm is considered to be successful only if it correctly finds all hyperedges of the hypergraph. On smaller examples, when the algorithm finds all hyperedges, it correctly computes all Walsh coefficients. When the algorithm fails (on this class of functions), it is most likely to fail when doing the order-2 probes. Thus, the formula of Theorem 9 should be applied with $j = 2$ and $k = 5$. The number of order-2 hyperedges is at most $50\binom{5}{2} = 500$ since there are $\binom{5}{2}$ order-2 hyperedges per subfunction. However, some of these overlap, and the actual number is about 420.

The algorithm of Section 8 was run for 1000 trials for each of $N = 40, 50, 60, 70, 80, 90$, where N is the number of probes per potential hyperedge. (i. e., N is the population size.) The algorithm was also run with the same parameters using randomly chosen backgrounds instead of backgrounds from a population. In addition, the first equation of Theorem 9 was solved for the success rate for the same values of N and with $j = 2$, $k = 5$, and $J = 420$. These are shown in the table on the left below. The table on the right shows the average number of function evaluations for these experiments.

These results suggest that when f is an embedded landscape with the number of subfunctions being $O(L)$, then the complexity is given by the formula of Theorem 9 even though populations were taken from a population rather than being randomly generated. Further theory and/or experiments are needed to confirm this hypothesis.

Accuracy			
N	Theory	Population	Random
40	0.1331	0.222	0.168
50	0.5889	0.658	0.648
60	0.8700	0.891	0.888
70	0.9640	0.963	0.967
80	0.9904	0.992	0.992
90	0.9975	1.00	0.995

Function Evaluations		
N	Population	Random
40	69008	279526
50	85889	345577
60	102547	411381
70	119241	490876
80	135907	540427
90	152501	604383

11.3 Solving some large problems

When the algorithm of Section 8 is run with function evaluation caching, memory size can be a limiting factor. To conserve memory, it is important that a bit-string encoding be used for background strings. On Intel 32-bit hardware, the maximum amount of memory that we could get from the Java virtual machine was about 1900 megabytes.

Under these limitations, the algorithm of Section 8 was able to find the structure of randomly generated 1000 bit 3-MAXSAT problems with 4300 clauses 8 times out of 10 using 24 trials per order-1 probe, 13 trials for order-2 probes, and 7 trials for higher order probes. The run time was about 2.4 hours per instance on a 2 GHz. machine, and the number of function evaluations 6,420,000 $\pm$ 2000.

Functions with randomly generated subfunctions are much easier. The algorithm of Section 8 was able to find the structure of 1200 bit problems with 150 randomly generated nonoverlapping subfunctions 100 times out of 100. The number of trials per probe was $12, 8, 6, 4, 3, 2, 2, 2$ for order $1, 2, 3, \ldots, 8$ probes respectively. The number of function evaluations per instance was 5,768,258, and the time was about an hour per instance on a 3 GHz. Pentium© 4.

12 Conclusions

The strength of the perturbational approach used in this paper is that a probe gives unambiguous information about the interaction of the variables involved in the probe that is not contaminated by noise from the interaction of other variables. Thus, probes can detect weak interactions between variables even when there are strong interactions between other variables. On the other hand, a probe only gives information on the relationship between those variables involved in the probe, and if there are many potential variable relationships to be tested, this means that many probes must be done.

This can be contrasted to methods that use either a random population or a population that results from executing some stages of an evolutionary computation algorithm. These methods are essentially using random or semirandom sampling to estimate the interaction effects of variables. Now the interaction effect of a specific collection of variables (the signal) is mixed up with the interaction effects of other variables (the noise), and a large population may

be needed to pick up the needed signal from the noise. Furthermore, when there are overlapping blocks, it may be very difficult to characterize this noise, and this makes it very difficult to give rigorous complexity bounds for this approach. However, these methods fit naturally into a population-based framework for optimization, and thus it is more evident how to combine a sampling-based linkage discovery with a sampling-based optimization.

This paper uses a very strict definition of what it means to successfully solve the linkage discovery problem. We say that the problem is solved only if all of the relevant hyperedges of the linkage hypergraph are successfully detected. Many papers (such as (Pelikan et al., 2000; Harik et al., 1999; Munetomo and Goldberg, 1999b)) on linkage use a weaker definition of a successful solution: A successful solution finds some fixed percentage of the blocks. Thus, as the problem size grows, there will be an increasing number of blocks that are not successfully detected. Further, in order to find a block where blocks are nonoverlapping, the algorithms in this paper only need to find enough edges in the linkage hypergraph to connect all of the vertices corresponding to that block.

There are two contributions of this paper. First, the paper gives a rigorous mathematical foundation for perturbational methods for determining the epistatic structure of a function from binary strings to the real numbers. These methods are closely related to the Walsh basis representation of the function.

Second, the paper gives two new randomized algorithms to solve the problem of detecting linkage (finding the components of nonlinearity) of a fitness function from fixed length binary strings to the reals. Both algorithms work as well on fitness functions with overlapping subfunctions (blocks) as they do on nonoverlapping subfunctions. The first algorithm generalizes the LINC algorithm of (Munetomo and Goldberg, 1999b) and (Munetomo and Goldberg, 1999a) to finding epistasis of arbitrary order. The primary parameter in the algorithm is the number of probes. If the function has k-bounded epistasis (is k-delineable in the terminology of (Munetomo and Goldberg, 1999a)), then rigorous bounds can be given for the number of probes that are needed, and this leads to a complexity analysis of the algorithm. The second algorithm generalizes the algorithms of (Kargupta and Park, 2001). This algorithm both determines the epistatic structure and finds the Walsh coefficients of a k-epistatic function. It is more practical when most of the Walsh coefficients of order less than k are zero. This algorithm is more efficient than the methods of (Kargupta and Park, 2001). A rigorous complexity analysis is given when the number of subfunctions grows linearly with the string length.

More research is needed in applying this class of algorithms to functions where the assumptions of k-bounded epistasis and sparseness of the Walsh basis representation are only approximately satisfied. Further research is also needed in understanding how these results can by used by genetic algorithms and estimation of distribution algorithms to take advantage of the epistatic structure of functions.

Acknowledgements

The authors would like to thank Michael Vose, Bill Derrick, and Mark Kayll for help with mathematical questions. In particular, the proof of Lemma 18 is due to Michael Vose.

References

Braunstein, A., Mezard, M., Weigt, M., and Zecchina, R. (2003). Constraint satisfaction by survey propagation. *arXiv.org:Condensed Matter Abstracts* http://arxiv.org/abs/cond-mat/0212451.

Brodie, E. D. (2000). *Why Evolutionary Genetics Doesn't Always Add Up*, pages 3–19. Oxford University Press, Oxford, England.

Chen, Y. P. and Goldberg, D. E. (2003). Tightness time for the linkage learning genetic algorithm. In et. al., E. C.-P., editor, *Genetic and Evolutionary Computation - GECCO 2003*, pages 837–849. Springer.

de Bonet, J. S., Isbell, Jr., C. L., and Viola, P. (1997). MIMIC: Finding optima by estimating probability densities. In et. al., M. C. M., editor, *Advances in Neural Information Processing Systems*, volume 9, page 424. MIT Press.

Deb, K. and Goldberg, D. E. (1992). Analyzing deception in trap functions. In *FOGA2*, pages 93–108, Palo Alto, CA. Morgan Kaufmann Publishers, Inc.

Gao, Y. (2003). Space complexity of estimation of distribution algorithms. Technical report, University of Alberta.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing, Co., Reading, MA.

Goldberg, D. E., Deb, K., Kargupta, H., and Harik, G. (1993). Rapid, accurate optimization of difficult optimization problems using fast messy genetic algorithms. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 56–64, San Mateo, California. Morgan Kaufman.

Harik, G. R. (1999). Linkage learning via probabilistic modeling in the ECGA. Technical report, University of Illinois. IlliGAL technical report 99010.

Harik, G. R., Cantú-Paz, E., Goldberg, D. E., and Miller, B. L. (1999). The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, 7(3):231–253.

Harik, G. R. and Goldberg, D. E. (2000). Learning linkage through probabilistic expression. *Computer Methods in Applied Mechanics and Engineering*, 186:295–310.

Heckendorn, R. B. (1999). *Walsh Analysis, Epistasis, and Optimization Problem Difficulty for Evolutionary Algorithms*. PhD thesis, Colorado State University, Department of Computer Science, Fort Collins, Colorado.

Heckendorn, R. B. (2002). Embedded landscapes. *Evolutionary Computation*, 10(4):345–376.

Heckendorn, R. B. and Whitley, D. (1999). Predicting epistasis from mathematical models. *Evolutionary Computation*, 7(1):69–101.

Hogg, T., Huberman, B. A., and Williams, C. P. (1996). Special issue SAT problems. *Artificial Intelligence*, 81(1-2).

Kargupta, H. (1996). The gene expression messy genetic algorithm. In *International Conference on Evolutionary Computation*, pages 814–819.

Kargupta, H. and Park, B. (2001). Gene expression and fast construction of distributed evolutionary representation. *Evolutionary Computation*, 9(1):43–69.

Kauffman, S. A. (1993). *The Origins of Order*. Oxford University Press, Oxford, England.

Mühlenbein, H. and Mahnig, T. (1999). Convergence theory and application of the factorized distribution algorithm. *Journal of Computing and Information Technology*, 7(1):19–32.

Mühlenbein, H., Mahnig, T., and Rodriguez, A. O. (1999). Schemata, distributions and graphical models in evolutionary optimization. *J. of Heuristics*, 5:215–247.

Munetomo, M. (2002a). Linkage identification based on epistasis measures to realize efficient genetic algorithms. In *Proceedings of the Congress on Evolutionary Computation (CEC) 2002*, pages 1332–1337. IEEE Press.

Munetomo, M. (2002b). Linkage identification with epistasis measures considering monotonicity conditions. In *Proceedings of the 4th Asia-Pacific conference on simulated annealing and learning (SEAL-02)*, pages 550–554.

Munetomo, M. and Goldberg, D. E. (1999a). Identifying linkage groups by nonlinearity/non-monotonicity detection. In et. al., W. B., editor, *Proc. of the Genetic and Evolutionary Computation Conference*, volume 1, pages 433–440, Palo Alto, CA. Morgan Kaufmann Publishers, Inc.

Munetomo, M. and Goldberg, D. E. (1999b). Linkage identification by non-monotonicity detection for overlapping functions. *Evolutionary Computation*, 7(4):377–398.

Niven, I. (1965). *Mathematics of Choice*. Mathematical Association of America.

Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley Publishing, Co.

Pelikan, M., Goldberg, D. E., and Cantú-Paz, E. (1999). BOA: The Bayesian optimization algorithm. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., iela, M. J., and Smith, R. E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume I, pages 525–532, Orlando, FL. Morgan Kaufmann Publishers, San Fransisco, CA.

Pelikan, M., Goldberg, D. E., and Cantú-Paz, E. (2000). Bayesian optimization algorithm, population sizing, and time to convergence. IlliGAL Report No. 2000001, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL.

Rana, S., Heckendorn, R. B., and Whitley, D. (1998). A tractable Walsh analysis of SAT and its implications for genetic algorithms. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 392–397, Menlo Park, CA. AAAI Press.