Efficient Linkage Discovery by Limited Probing

Robert B. Heckendorn[1] and Alden H. Wright[2]

[1] University of Idaho, Moscow, ID 83844-1010 USA, heckendo@cs.uidaho.edu,
[2] Computer Science, University of Montana, Missoula, MT USA, wright@cs.umt.edu

Abstract. 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 Introduction

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, however, are not nearly as simple. For many, the contribution of a bit in the domain to the function value 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*" [Bro00].

Applied to the case of evolutionary computation the "units" in the quote above refer 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 [HW99]. High epistasis, however, is no guarantee of a difficult problem. Nor is low epistasis a guarantee of an easy problem. In fact, MAX3SAT problems are equivalent to problems of low epistasis in which all epistatic interactions are known and they are provably NP-complete [Hec99]. 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 [MG99b,MG99a,KP01]. 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 [MM99] also suggest

E. Cantú-Paz et al. (Eds.): GECCO 2003, LNCS 2723, pp. 1003–1014, 2003.

applying the UMDA algorithm where each component makes up a string position with a higher-order alphabet. Mühlenbein, Mahnig, and Rodriguez [MMR99] give a factorized distribution algorithm (FDA) that applies to additively decomposed functions (that we call embedded landscapes in this paper).

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 [MG99b] [MG99a]. These papers also summarize some other approaches to the problem. Kargupta et al. [KP01] 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 in this paper are **blackbox 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. This algorithm makes much more efficient use of function evaluations than previous algorithms.

2 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. Since the L-bit binary representations of the integers in the interval $[0, 2^L)$ coincide with the elements of $\mathcal{B}$, a bit strings 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 $|\mathcal{B}_m| = 2^{bc(m)}$. Square brackets are used to denote an **indicator function**: if $expr$ is an expression that may be true or false, then

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

3 Walsh Analysis and Embedded Landscapes

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 i^{th} Walsh function is defined:

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

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 [Hec02]:

Theorem 1. *(Embedded 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.

4 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. 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 $c \in \mathcal{B}_{\overline{m}}$. c is called the **background** of 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.

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.

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

By the Balanced Sum Theorem for Hyperplanes [HW99] 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 without further information we can say very little. The following theorem identifies the set of Walsh coefficients.

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

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

Proof. Recall that $f = \sum_{j \in \mathcal{B}} w_j \psi_j$. Thus,

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

□

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 1. *(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 4. *(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 5. *(Nonzero Probe Existence) Given a maximal nonzero Walsh coefficient w_m, for all a: $a \subseteq m$, 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 $P(f, m \oplus n, i \oplus c) = P(f, a, i \oplus c) \neq 0$. □

5 The Linkage Graph and Hypergraph

A **hypergraph** is a collection of vertices V together with a family of subsets E of V called hyperedges where each hyperedge is nonempty. The set of vertices of the **linkage hypergraph** is the set of string positions j. A set of vertices corresponding to mask $m \neq 0$ is a hyperedge if there is a $c \in \mathcal{B}_{\overline{m}}$ such that $P(f, m, c) \neq 0$. A hyperedge will be identified with the corresponding mask. The **order of a hyperedge** is the number of ones in the mask. In view of Theorem 5, 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 2. *If m is a hyperedge of the hypergraph, and if $a \subseteq m$, then a is also a hyperedge.*

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

The order-j linkage detection algorithm 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 [MG99b]. 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.

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 its 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.

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.

Theorem 6. *(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 5 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 6 cannot be improved under these assumptions. 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 7. *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*

$$N \geq \frac{\ln(1-\delta^{1/J})}{\ln(1-2^{j-k})}$$

or

$$N \geq -2^{k-j}\ln(1-\delta^{1/J})$$

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 6 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 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, note that $-\ln(1-x) \geq x$ for any $x > 0$ from the Taylor series of $-\ln(1-x)$. Thus, $2^{k-j} \geq 1/\ln(1-2^{j-k})$. □

Strictly speaking, these results do not apply to the LINC algorithm of [MG99b] since the above analysis assumes that the backgrounds of probes are chosen independently, and this is not the case for the LINC algorithm. However, our empirical results show that these formulas are quite accurate when the backgrounds are chosen from a population. (See Section 8.) In fact, it is much more accurate than the population sizing formula given by [MG99b].

Thus, the overall worst-case complexity of the order-2 linkage detection algorithm is $O(2^k L^2 \ln(1-\delta^{1/J}))$ if one wants to maintain the same probability of overall success as the string length increases. Munetomo and Goldberg [MG99b] give the overall complexity of their LINC algorithm as $O(2^k L^2)$, but this assumes that the probability of success per subfunction of the embedded landscape stays constant as the string length increases, which would seem to be a less desirable assumption.

6 Computing the Walsh Coefficients Using the Kargupta-Park Top-Down Algorithm

Kargupta and Park [KP01] 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

$$P(f, j, 0) = w_j + \sum_{j \subset u} w_u \psi_u(0) = w_j + \sum_{j \subset u} w_u \tag{2}$$

(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 [KP01].)

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

7 Detecting Linkage and Computing the Walsh Coefficients

Kargupta and Park [KP01] 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 6.

In this section, we give a well-specified algorithm that efficiently finds the nonzero Walsh coefficients and computes their values. The algorithm 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 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 8 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.

Theorem 8. *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:

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

Proof. Using the definition of a probe, the conclusion can be rewritten as:

$$f(m \oplus c) = \sum_{a \in \mathcal{B}_m} (-1)^{bc(a)} \sum_{i \in \mathcal{B}_a} (-1)^{bc(i)} f(i \oplus c)$$

We prove this by induction on $bc(m)$. The base cases of $bc(m) = 0$ and $bc(m) = 1$ are easy. Let $m = u \oplus v$ where $u \wedge v = 0$, $u \neq 0$, $v \neq 0$. Then

$$\begin{aligned}
&\sum_{a \in \mathcal{B}_m} (-1)^{bc(a)} \sum_{i \in \mathcal{B}_a} (-1)^{bc(i)} f(i \oplus c) \\
&= \sum_{j \in \mathcal{B}_u} (-1)^{bc(j)} \sum_{k \in \mathcal{B}_v} (-1)^{bc(k)} \sum_{i \in \mathcal{B}_{j \oplus k}} (-1)^{bc(i)} f(i \oplus c) \\
&= \sum_{j \in \mathcal{B}_u} (-1)^{bc(j)} \sum_{r \in \mathcal{B}_j} (-1)^{bc(r)} \sum_{k \in \mathcal{B}_v} (-1)^{bc(k)} \sum_{s \in \mathcal{B}_k} (-1)^{bc(s)} f(s \oplus r \oplus c) \\
&= \sum_{j \in \mathcal{B}_u} (-1)^{bc(j)} \sum_{r \in \mathcal{B}_j} (-1)^{bc(r)} f(v \oplus r \oplus c) \\
&= f(u \oplus v \oplus c) = f(m \oplus c)
\end{aligned}$$

□

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.

The basic idea of the bottom-up part of the algorithm (TRAVERSE-HYPERGRAPH) is to do a breadth-first traversal of the lattice of masks, starting with the empty mask, then looking at the order-1 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.

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 probe value 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.

The TESTBYPROBES function does up to N probes using the mask a. 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 $null$.

```
TRAVERSE-HYPERGRAPH()
    population.initialize()
    hypergraphList.initialize()
    queue.initialize()
    m ← { }          // Empty mask
    ProbeValue ←TESTBYPROBES(a)
    if ProbeValue ≠ null then
        queue.add(m)
        hypergraph[m] ← ProbeValue
    end if
    while queue.notEmpty() do
        m ← queue.remove()
        probeValue ← hypergraph[m]
        for all supersets a of m of cardinality bc(m) + 1 do
            if all subsets of a of cardinality bc(m) are in the hypergraphList then
                ProbeValue ←TESTBYPROBES(a)
                if ProbeValue ≠ null then
                    queue.add(a)
                    hypergraph[a] ← ProbeValue
                    hypergraphList.addFirst(a)
                end if
            end if
        end for
    end while
```

The top-down part of the algorithm (COMPUTE-WALSH-COEFS) 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.

The algorithm is based on Equation 2. 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 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 ∈ hypergraphList do   //traverse in the reverse order from the 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

8 Empirical Results

The test function used in this section is an embedded landscape with 50 5-bit subfunctions and a string length of 50. Each subfunction is 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 maximum value given by the linear function. The support of each subfunction is randomly chosen. A value is considered to be zero if it is less than 10^{-7}. This class of test function represents the worst case for the algorithms given in this paper.

The algorithm used is that given in Section 7. 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 7 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 7 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 7 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 7. 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

9 Conclusions

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. The first generalizes the LINC algorithm of [MG99b] and [MG99a] 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 [MG99a]), 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 [KP01]. 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 [KP01].

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 genetic algorithms and estimation of distribution algorithms can work in tandem to take advantage of the epistatic structure of functions that can be discovered by algorithms such as the ones given in this paper.

References

[Bro00] E. D. Brodie. *Why Evolutionary Genetics Doesn't Always Add Up*, pages 3–19. Oxford University Press, Oxford, England, 2000.

[Hec99] Robert B. Heckendorn. *Walsh Analysis, Epistasis, and Optimization Problem Difficulty for Evolutionary Algorithms*. PhD thesis, Colorado State University, Department of Computer Science, Fort Collins, Colorado, 1999.

[Hec02] Robert B. Heckendorn. Embedded landscapes. *Evolutionary Computation*, 10(4): 345–376, 2002.

[HW99] R. B. Heckendorn and Darrell Whitley. Predicting epistasis from mathematical models. *Evolutionary Computation*, 7(1):69–101, 1999.

[KP01] H. Kargupta and B. Park. Gene expression and fast construction of distributed evolutionary representation. *Evolutionary Computation*, 9(1):43–69, 2001.

[MG99a] Masaharu Munetomo and David E. Goldberg. Identifying linkage groups by nonlinearity/non-monotonicity detection. In W. Banzhaff et. al., editor, *Proc. of the Genetic and Evolutionary Computation Conference*, volume 1, pages 433–440, Palo Alto, CA, 1999. Morgan Kaufmann Publishers, Inc.

[MG99b] Masaharu Munetomo and David E. Goldberg. Linkage identification by non-monotonicity detection for overlapping functions. *Evolutionary Computation*, 7(4): 377–398, 1999.

[MM99] Heinz Mühlenbein and Thilo Mahnig. Convergence theory and application of the factorized distribution algorithm. *Journal of Computing and Information Technology*, 7(1):19–32, 1999.

[MMR99] Heinz Mühlenbein, Thilo Mahnig, and Aberto O. Rodriguez. Schemata, distributions and graphical models in evolutionary optimization. *J. of Heuristics*, 5:215–247, 1999.