INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

NOTE TO USERS

This reproduction is the best copy available.

UMI

DISSERTATION

WALSH ANALYSIS, EPISTASIS, AND OPTIMIZATION PROBLEM DIFFICULTY FOR EVOLUTIONARY ALGORITHMS

Submitted by

Robert B. Heckendorn

Department of Computer Science

In partial fulfillment of the requirements

for the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 1999

COLORADO STATE UNIVERSITY

July 4, 1999

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY ROBERT B. HECKENDORN ENTITLED WALSH ANALYSIS, EPISTASIS, AND OPTIMIZATION PROBLEM DIFFICULTY FOR EVOLUTIONARY ALGORITHMS BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Michael Kirby

Adele E. Howe

D Whitley

Adviser

Department Head

ABSTRACT OF DISSERTATION

WALSH ANALYSIS, EPISTASIS, AND OPTIMIZATION PROBLEM DIFFICULTY FOR EVOLUTIONARY ALGORITHMS

The epistasis of a function is the bitwise nonlinearity of a function whose domain is the set of bit strings of length L. Epistasis is related to problem difficulty for evolutionary algorithms. Walsh analysis can be used to quantify epistasis. In this dissertation Walsh analysis is developed in detail starting with the definitions for the Walsh transform.

The epistasis of a polynomial is shown to be bounded by the degree of the polynomial and the epistasis of logical expressions by the number of variables involved. The effects on epistasis of problem representation operators such as: bit extraction, scaling, translating, and Gray code are also studied. The epistasis of functions composed of subfunctions combined by various operators is predicted. I show that functions can display odd and even parity and prove several theorems on the invariance of these properties. An application of these theorems demonstrates that picking the proper problem representation reduces epistasis, often making the problem easier to solve. Several new measures of epistasis are created including: Walsh sums, Walsh counts, function order, and coverage. New and useful mathematical tools for Walsh analysis are presented such as *pack*, *unpack*, and spectral functions and hyperplane numbering.

The concept of embedded landscapes is introduced. An embedded landscape is a sum of a set of subfunctions each of which takes as its argument a subset of bits from the domain of the landscape. Embedded landscapes with bounded subfunction domains are shown to have epistasis limited to that of the subfunctions. Both MAXkSAT and NK-landscapes are shown to be embedded landscapes. I prove that all the Walsh coefficients of an embedded

landscape. composed of subfunctions of bounded domain. can be computed in polynomial time. Summary statistics (variance. skew. etc.) can also be computed in polynomial time. I conclude that even though all Walsh coefficients of a function can be known in polynomial time. the function can still be NP-complete.

Robert B. Heckendorn
Department of Computer Science
Colorado State University
Fort Collins. Colorado 80523
Summer 1999

PREFACE

This work serves not only as a dissertation but as a general reference work on the application of Walsh analysis to optimization problem landscapes. With that in mind, I have tried to be thorough in my presentation, methodical in proof, and provide a detailed index. Some other features of this work are:

- Key ideas and terms are written in bold to alert the reader of their importance and so that the reader can quickly refer back to recent terms they might have forgotten.
- Italics is used for emphasis and a typewriter font for computer code.
- Often when an important new mathematical idea is introduced, it will be followed by a section labeled "Observations". This section quickly lists a series of mini-theorems or identities that require but a moments thought to prove and provide an excellent review of concepts. The observations sections also make a convenient point of reference for identities when reading later proofs.
- Theorems are all named mnemonically and referred to by these names. This should reduce the amount of page turning to find out what theorem number N was. However, should you have to refer back to a theorem, the names of all theorems can be found in the index.

- A few of the earlier theorems are not original but were reproven by me and are included for completeness and uniformity of approach.
- A list of symbols used is provided after the table of contents.
- A section devoted to notation and terminology can be found at the end of the introduction. It is provided at that point to encourage the reader to become familiar with the terms and notations used before they are encountered.
- Sections which contain empirical work all have titles that begin "Experiment" to distinguish them from theoretical sections.

Although not all of the theorems are original, for uniformity of treatment, all proofs in this work are my work and use a common set of notation. Some theorems have analogues in other fields and similar theorems have been proven there. For example, some theorems are similar to those found in discrete Fourier analysis. However, discrete Fourier analysis does not consider domains to consist of bit strings or subsets of bit strings (Weaver, 1989). Harmonic analysis is another area where similar work has been done. But here again, the work has been related more to signal processing and circuit reduction than in fitness function analysis (Lechner, 1971; Bachman, 1964). The contribution of others has been attributed where possible. I will point out here that Dr. Darrell Whitley was a major contributor to the NP-completeness proofs.

I would like to thank my advisor Dr. Darrell Whitley of Colorado State University who has been an endless source of encouragement and calm throughout my PhD work. He has shown me how publishing should be done. My thanks also to Soraya Rana who has a gift for empirical analysis and an uncanny skill at pulling patterns from data. Her empirical contribution to our joint papers has been invaluable. My thanks also to Dr. Adele Howe who has provided her useful insights not only into proper statistical method, but into the long term view of a career in academia. My thanks to the rest of the committee: Dr. Michael Kirby of the Math Department, Dr. Ross Beveridge, and Dr. Wim Bohm of the Computer Science Department for their constructive suggestions in reviewing this document. My gratitude to Larry Pyeatt for his dissertation macros for LaTex. Thanks also

to Jack Applin for being a friend through my departure from Hewlett-Packard and years back at school. Finally, I owe an eternal debt to my wife, Marilyn, for her tolerance on the numerous occasions when she was tired of my work on this document and for supporting me when I was tired of it.

Robert B. Heckendorn
Fort Collins, Colorado, USA
July 10, 1999

DEDICATION

To my mother who taught me the love of books and
who always wanted me to get a PhD.

and

To Marilyn who is the love and focus of my life.

and

To Abbey the corgi who showed me how much
we can learn about life and love from another species.

"Walshing Matilda.
Walshing Matilda.
You'll come a Walshing Matilda with me..."

– An orthogonal projection of
an Australian folk song

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

$\alpha(h)$ Fixed bit positions in hyperplane h. page 20

$\langle f \rangle_h$ Average of function f over hyperplane h. page 20

$\wedge$ Logical AND . page 19

$A(p)$ Evolutionary algorithm with parameters p. page 4

$h_{m,n}$ Hyperplane numbering. page 73

$\mathcal{E}(g, m)$ Embedding of function g using mask m. page 78

$\beta(h)$ The 1 bit positions in hyperplane h. page 20

$\mathcal{B}$ Bit space . page 18

$\mathcal{B}^L$ Bit space - L bit space. page 18

$bc(\mathbf{i})$ Bitcount - Counts the number of ones in i. page 19

$C(m, n)$ Conjunctive clause defined by two masks. page 136

$f \circ g$ Convolution operator. page 44

$i \subseteq j$ Containment - Bit string i is contained in j. page 19

$dim(f)$ Dimension of a function. page 19

$D(m.n)$ Disjunctive clause defined by two masks. page 136

$f \cdot g$ Dot product. page 53

f_{decode} Decoding function. page 3

$f_{\text{desirability}}$ Desirability function. page 3

$\bigoplus_{i=0}^{L-1}$ Iterated EXCLUSIVE-OR . page 27

x[i, j] Extract bits i through j from string x. page 19

$\ominus$ Logical EXCLUSIVE-OR . page 19

f_{fitness} Fitness function. page 3

$\mathcal{K}_b$ Walsh count. page 89

$\tilde{\mu}_r(h)$ Moment: The r^{th} moment of hyperplane h about the mean of the hyperplane. page 175

μ Mean of the function. page 171

μ_r Moment: r^{th} moment about the mean of the function. page 171

f_{model} Model function. page 3

$\overline{x}$ Logical negation . page 19

$\Omega(f)$ Order of function f. page 90

$\vee$ Logical OR . page 19

$o(\pi)$ Order of partition π. page 20

$o(h)$ Order of hyperplane h . page 20

$\mathbb{R}$ Reals - The real numbers. page 18

$\lfloor n \rfloor_k$ Rounddown - Largest integer less than or equal to n divisible by k. page 18

$\xrightarrow{\text{shift}}$ Shift bits right. page 104

$\sum_{k\,:\,\iota=k\wedge m}$ Sum over conditional - The sum over all k such that $i = k \wedge m$. page 18

ψ_i Walsh function - The i^{th} Walsh function. page 26

W_b Walsh sum. page 90

w_j Walsh Coefficient - The j^{th} Walsh coefficient. page 27

$X(m, n)$ EXCLUSIVE-OR clause defined by two masks. page 140

$Y(x)$ Y Function - Maps 0, 1 of parity to $+1, -1$ of Walsh functions. page 27

$\mathcal{Y}$ Signed unit space . page 18

$\mathbb{Z}_k$ Mod integers - represents the nonnegative integers mod k. page 18

$\mathbb{Z}$ Nonnegative integers - The nonnegative integers. page 18

Chapter 1

Introduction

These days the operative word in industry is optimization. However, many of the practical optimization problems faced by these companies have mind bogglingly huge search spaces. These problems often do not succumb to classic operations research approaches. However, the payoff for finding good answers, that might not even be optimal, is so large that companies are relying on innovative approaches to optimization to get the job done. For example, if a company has a billion dollar manufacturing process that can be improved by 5%, then they can save $50 million.

With the advent of cheap, fast computational hardware, a very successful approach to optimization is to borrow techniques from nature. The resulting approaches include simulated annealing (Kirkpatrick et al., 1983), genetic algorithms (Goldberg, 1989c), evolutionary strategies (Bäck et al., 1991) and genetic programming (Koza, 1992). A subclass of these "natural" algorithms is evolutionary optimization algorithms. They are stochastic optimization algorithms which maintain a set of potential solutions, also referred to as a population, from which new potential solutions are generated.

Genetic algorithms and evolution strategies fall into this category. Evolutionary algorithms have been applied to real world optimization problems and found to be competitive with more classical approaches. For example, the General Electric engines on the Boeing 777 have turbine geometry designed by an evolutionary algorithm (Petit, 1998). John Deere and Volvo both use evolutionary algorithms to adaptively schedule the assembly of complex farm machinery and tractor trailer trucks (Rao, 1998; Petit, 1998). Some other examples of problems successfully tackled by evolutionary methods are cryptanalysis (Bagnall et al.,

1997). car suspension design (Deb and Saxena. 1997). telecommunications network planning (Brittain et al.. 1997). layout of containers for containership loading (Todd and Sen. 1997). and knowledge discovery in databases (Venturini et al.. 1997). All of these difficult and important optimization problems tend to involve complex nonlinear mathematical and/or combinatorial models. They are often multiobjective discontinuous optimization problems where the best known algorithms for exact solutions are exponential in the size of the problem.

Unfortunately. optimization by evolution based algorithms is still very much an art. Given a problem. practitioners have little intuition about how to choose the best internal representation for the problem or which of the large set of evolutionary algorithms works best on a given class of problems. Often researchers will implement several algorithms and see which one performs best. This indicates a lack of understanding of what makes a problem difficult for a given evolutionary algorithm. Whitley et al. (Whitley et al.. 1995b) point out that this has even lead to test problems being created without the designers knowing how difficult the problems are or how the difficulty changes as the problem is scaled to higher dimensions.

There are also deeper questions involved. Stuart Kauffman examines evolution from a biological point of view in his book *At Home in the Universe* (Kauffman. 1995). There he argues that evolutionary optimization naturally occurs and that the structure of the biological and chemical search spaces found in nature are such that it allows evolution to work well.

It is surprising that such a simple process works so well in both natural industrial settings. Is it because the process is inherently extremely powerful or is it because many of the problems from the real world are. in some underlying sense. easy to solve using this approach? What properties of real world problems tend to make the problems amenable to evolutionary approaches? Can those properties be easily detected? Can problems then be made less difficult by altering them to reduce undesirable properties? This dissertation is about understanding the relation between problem difficulty for evolutionary algorithms and structural features of problem. This chapter provides the background and limits the

scope for this research. Problems. algorithms. and difficulty are each discussed in detail including key terminology and examples.

1.1 The Optimization Problem

Most optimization problems tackled by evolutionary algorithms can be modeled as a composition of three functions:

$$f_{\text{fitness}} = f_{\text{desirability}}(f_{\text{model}}(f_{\text{decode}})) : \mathcal{B}^L \rightarrow \Re \tag{1.1}$$

Potential solutions are encoded in an L bit string. The solution space to be searched is now the L bit space denoted $\mathcal{B}^L$. The **decoding function**. f_{decode}. takes a potential solution as a bit string and converts it to a data structure that is understandable by the mathematical/computational model of the problem to be optimized. For example. for a Traveling Salesperson Problem(TSP) problem. it may convert the bit string into a list of cities representing a tour. The **model function**. f_{model}. takes a decoded potential solution and models the problem. The function returns a set of features that can then be judged for quality. Using our TSP example. f_{model} might take a list of cities and return the length of the tour which could then be judged for quality. Finally. the desirability of the modeled features is rated on a real numeric scale by a **desirability function**. $f_{\text{desirability}}$. which can then be maximized. In contrast to the model function. which makes no qualitative judgement on the proposed solution. the desirability function is the automated estimate of the worth the user places on the potential solution. Desirability is usually measured as number such as a real. since the desirability must allow for ranking of results. The importance of the desirability function is especially apparent in multiobjective functions. For example: if f_{model} describes some operational features of a factory. $f_{\text{desirability}}$ might have to evaluate the simultaneous desirability of product quantity and quality. amount of pollution. and resource consumption. The result of the composition is a **black box evaluation function**. f_{fitness}. that evaluates desirability of L bit strings representing potential solutions. It is important to see that all three factors of decoding. modeling. and desirability are formulated in the

single function, f_{fitness}. Although it is sometimes useful to consider the affects of different encodings separately, the definition I present for f_{fitness} includes the encoding.

Borrowing from biology, the evaluation function for evolutionary algorithms is called the **fitness function** and the encoded bit string is a **chromosome**. A set of chromosomes is referred to as a **population**. We can now envision the optimization process as an evolutionary process using a population of chromosomes to try to evolve a highly fit chromosome and hence a "good" answer.

The process of finding a fittest chromosome can be mathematically expressed as: given a domain $\mathcal{D}$ and a real valued function $f : \mathcal{D} \rightarrow \Re$, find $x \in \mathcal{D}$ such that $f(x) \geq f(y)\ \forall\ y \in \mathcal{D}$. In our case, $\mathcal{D}$ is the set of allowable chromosomes and f is the fitness function. This is simply a **classical optimization problem** and the value of x is a **global optimum**. There may be many global optima, but in this research I will be concerned with only discovering one. Also I will, without loss of generality, only be concerned with maximizing the function.

For my empirical work, I will require the algorithm find the global optimum. This will force the tested algorithms to include the conditions of execution that will be found as the algorithm approaches the global optimum. This means I must use problems where the global optimum can be computed. For problems where analysis fails, the global optimum must be found by exhaustively searching the domain. The later case will require problems to be fairly small in size. For current hardware that is less that 25 bits.

1.2 Evolutionary Algorithms

A **stochastic optimization algorithm** is an optimization algorithm that uses a nondeterministic process to sample function values, $f(x)$, $x \in \mathcal{D}$, in order to attempt to find a global optimum for f. From an implementation standpoint, a specific evolutionary algorithm is denoted $A(p)$ where A represents the fixed process description and p the parameters fixed for a particular instance of the algorithm $A(p)$. In this section, I will review the key attributes and types of evolutionary algorithms, provide an outline for a simple class of evolutionary algorithms, and discuss the terminology and tenets of their mechanics.

An **evolutionary optimization algorithm** is an evolutionary algorithm modeled after

the evolutionary processes we find in nature. All evolutionary algorithms generally:

- model the solution space as a string of bits called a *chromosome*.
- maintain a population of chromosomes.
- have a black box fitness function that evaluates the quality of the chromosomes.
- apply mutation and/or crossover like operators to generate new chromosomes.
- apply selective/competitive pressure to force general improvement in the fitness of the maintained population.

```
initializePopulation();
while ( not goodEnough() ) {
      selectByFitness(a, b);
      child = mate(a, b);
      mutate(child);
      insertByCompetition(child);
}
```

Figure 1.1: General Outline of an Evolutionary Algorithm

Many evolutionary algorithms follow the algorithm outlined in Figure 1.1. Other less detailed variants of this outline can be found in Schwefel (Schwefel and Bäck, 1998) and in Mitchell (Mitchell et al., 1992). The outline employs six algorithmic component functions. These functions constitute, in part, the parameters p in $A(p)$. The first function, `initializePopulation`, initializes the population of chromosomes. As the algorithm loops in the while loop it will generally seek to improve the fitness of the best chromosome in the population. The second function, `goodEnough`, tests if the best fitness of the chromosomes in the population, as computed by f_{fitness} above, meets some standard set by user. The function `selectByFitness` selects two chromosomes from the population for use in generating a new potential solution.

Searching is generally directed by generating a new potential solutions in at least two ways. The `mate` component performs a blending of the two chromosomes, much as crossover of biological chromosomes blends to chromosomes. `mutate` alters the chromosome, generally

in small ways. These two search operators may or may not use knowledge of the encoding function to attempt to increase fitness of the resulting new potential solutions. For example `mutate` may only modify bits that incrementally effect the output of the model function.

Once a new chromosome is generated it must be decided if the chromosome should be inserted into the population. The function `insertByCompetition` usually does this by some form of fitness based competition with chromosomes in the population.

Even though this is a simplified outline of an evolutionary algorithm it is sufficient to model many popular evolutionary algorithms. For example, a population based random walk can be modeled by setting `mate` to be the identity, `mutate` to flip a single bit of the chromosome, and `insertByCompetition` to randomly replace an element in the population. By changing the `insertByCompetition` to replace the worst element in the population we get, a population based random bit climbing (without restarts) (Whitley et al., 1995b). Simulated annealing (Kirkpatrick et al., 1983) has this basic form as well if varying degrees of mutation are allowed based on a cooling schedule.

The algorithmic outline in Figure 1.1 represents a class of evolutionary algorithms called **steady state genetic algorithms** in that the population elements are modified continually one chromosome at a time (Whitley, 1989; Davis, 1991). Another popular evolutionary algorithm is the **generational genetic algorithm** such as the popular simple genetic algorithm (SGA) (Goldberg, 1989c). These are algorithms in which the whole population is used to generate an entirely new population and most or all of the old population is replaced all at once a new one. The successive populations are referred to as **generations**.

In order to understand how evolutionary algorithms work, it is important to understand the two basic tenets of evolutionary computation. In general, evolutionary algorithms work by balancing **exploration** with **exploitation** in selection of new chromosomes (Mitchell, 1996; Kauffman, 1993) and using selection and competition pressure to push a population to improved fitness (Holland, 1975).

Balancing exploration and exploitation can be thought of in terms of an ant colony. If the ant colony is only sending out scouts to find new food, but never goes and collects the food, it starves. This is like a stochastic search algorithm that never explores radically new

domain values may never explore places that may lead to a global optimum. If the ant colony only collects food from known sources. the food may be exhausted and the colony starves. This is like the stochastic search algorithm that only hill climbs to an optimum. With out exploration it is likely to be stuck in a local optimum.

Our simple algorithm demonstrates three important aspects of search that are characteristic of evolutionary algorithms.

- **Global search** selects new random chromosomes in the search space or large regions of the search space. It's exploratory nature has the advantage of occasionally adding fresh regions of highly fit chromosomes to the population. It has the disadvantage of not being able to exploit any information in the current population on where fit chromosomes lie. In our model. global search only occurs in the `initializePopulation` routine. Some evolutionary algorithms acquire the advantages of global search by reinitializing the population. also called restarts (Maresky et al.. 1995: Tsutsui et al.. 1997). Another approach is to use long jumps or massive mutation which. Kauffman defines as jumping "beyond the correlation length of the space" (Kauffman. 1995). to effectively get a global search. The CHC algorithm (Eshelman and Schaffer. 1991: Eshelman. 1991) uses constrained restarts in the form of catastrophic mutation which entailed cloning the best chromosome throughout the population and flipping 35% of the bits in all of the clones.

- **Local search** selects new chromosomes that are "near" chromosomes known to be fit. The term local search refers to the neighborhood concept that for a given chromosome there is a local neighborhood of chromosomes "around" the given chromosome with similar fitness. Progress is made if the function is well correlated with respect to the neighborhood of search. but is hampered by undesirable local minima with respect to that neighborhood. The `mutate` routine in our algorithm creates new chromosomes by taking a copy of chromosome in the current population and altering it slightly.

- **Piecewise search** - Both the previous modes of search are based on modifying at most a single chromosome. This kind of search can be found in a random bit climbing algorithm. Piecewise search takes its cue from sex in biology and is characteristic of a

genetic algorithm. Piecewise search, as its names implies, assembles a new chromosome from two or more chromosomes in the population much like crossover of chromosomes in nature. The idea being that perhaps the new chromosome will get the best parts of its parent's chromosomes. Piecewise search is performed by `mate` in our model.

1.3 Problem Difficulty

An evolutionary algorithm works by probing the function. The number of fitness function evaluations used in finding a solution is representative of the amount of information gathered in order to reach a solution. This supports the practical view that since fitness function evaluations tend to dominate the execution time of the program they should be used as a measure of difficulty. Therefore, the **execution time** of evolutionary algorithm $A(p)$ on problem f is measured in terms of the number of evaluations of f that are made before termination and is denoted $N(A(p), f) \in \mathbb{Z}$. Most algorithms employ a maximum number of function evaluations after which the algorithm will terminate. Some try to determine when progress is no longer being made and terminate or restart. Even though this was not explicitly stated in the algorithm outline of Figure 1.1, for empirical purposes, I will use the maximal number of evaluations approach.

If A were deterministic, then $N(A(p), f)$ would be a fixed value, but since A is stochastic, it is, in fact, a random variable. Therefore a sample distribution of N's for a fixed $A(p)$ and f may serve as an estimation of performance. Another way to view this is that N is an indicator of the difficulty of the optimization problem f for algorithm $A(p)$. Since N tends to reflect actual execution time, my research focuses on the relationship between algorithms $A(p)$, problems f and difficulty $N(A(p), f)$.

The difficulty of a problem is inextricably tied to the algorithm applied to solve the problem and the efficacy of an algorithm is tied to the problems to which the algorithm is applied. At first, this seems counterintuitive. Surely we can say that regardless of the problem random bit climbing is, in general, a better algorithm than just random guessing! Surprisingly, this is not the case. Wolpert and Macready (Wolpert and Macready, 1997) showed that, in fact, no algorithm is better than any other, where better is measured by

the number of distinct points examined in the search for the best answer, measured over *all* possible problems of finite domain. By a duality argument, they show no problem is any more difficult than another if measured against all possible algorithms. This is because, in the end, for any given problem, slow algorithms and fast algorithms cancel to a common mediocrity. Or as Wolpert and Macready put it (Wolpert and Macready, 1997), "if an algorithm performs well on a certain class of problems then it necessarily pays for that with a degraded performance on the set for all remaining problems." This means that for an algorithm that randomly guesses, there exist problems which it guesses the answer right away while the random bit climber must search for the answer. These important results are called the **No Free Lunch Theorems (NFL)**.

The NFL results prevent us from definitively declaring any problem easier or harder for all algorithms, but if I select the measure of difficulty as the number of points explored using a *specific class of algorithms*, I am no longer forced to claim that all problems are of equal difficulty. With this in mind, this dissertation will concern itself with various classes of problems and determining their difficulty.

1.4 Structure, Problem, and Difficulty

So far in this chapter I have discussed algorithms, problems, and measuring difficulty. The goal of this research is to associate the **structure** of the **problems** with the resulting **difficulty** with respect to evolutionary algorithms **algorithms**. By **Structure** I mean the orderly composition of the problem in terms of specific features. If the structure can be measured or if problems, by design, are known to contain specific structure then perhaps I can find a causal relationship between structure and difficulty. By associating structural features with difficulty, I may learn more about the mechanisms of problem difficulty. I may then be able to mitigate the effects of these mechanisms for that class of problems and algorithms. In the next section I will discuss epistasis, the primary kind of structure I will be exploring, and briefly talk about other kinds of important structure.

1.4.1 Epistasis

Epistasis is defined by looking at a function as a sum of contributing epistatic factors, one for each possible subset of interacting bits. This leads to this definition of epistasis I have adapted from biology: **epistasis** is that part of the function value that cannot be attributed to a weighted sum of the individual bit values (the linear effects), i.e., it is the nonlinear interactions between bits. But this biological definition does not let one speak of the single bit interactions, or look at the complete function as a sum of epistatic values. For uniformity of mathematical analysis, in this dissertation I will enlarge this definition of epistasis to include the separate linear effects, that is, the contribution of single bits to the function value. I will also include the constant term for no bits interacting.

The best explanation of epistasis is an intuitive one by analogy. Which offers better security for your bike? Four 1 digit bike locks or one 4 digit bike lock? In the first case each lock is independent and can be solved in series. If each 1 digit lock has only 10 states then the four locks can be opened in at most $4 * 10 = 40$ trials. In the second case, opening of the lock requires that all 4 digits be set before the lock opens. This may require as many as $10^4 = 10000$ trials because all of the digits are **interdependent**. Unlike the four lock case, the interdependence of the digits means that knowing any subset of digits tells us nothing about whether we are closer to opening the lock. The same can be true when searching a bit string for the optimal configuration. A function over a string with a **high epistasis** has a large number of interdependent bits. They all must be set correctly to obtain the optimum value for the function. Knowing anything about a subset of the interacting bits tells us nothing about whether we are closer to the optimum. Bits that are interdependent are said to interact epistatically. If a problem is **highly epistatic**, it has large numbers of bits in the domain that are interdependent. Functions with nonlinear epistasis are called **bitwise nonlinear functions**. When, for a function $f : \mathcal{B}^L \rightarrow \mathbb{R}$, all possible 2^L subsets of bits epistatically contribute to the function value, the function is said to be **fully epistatic**.

It seems important to select epistasis as a focus of study for two reasons. First, it seems reasonable to hypothesize that large subsets of bits which epistatically interact must, like with the bike lock analogy, be considered simultaneously when optimizing. This would tend

to make the optimization task more difficult. In fact it has been argued in the literature, by Davidor (Davidor, 1991) and Reeves and Wright (Reeves and Wright, 1995b; Reeves and Wright, 1995a), and others (Heckendorn et al., 1997; Voget, 1995), that the distribution and level of epistasis is often a predictor of the difficulty of an optimization problem.

Secondly, Goldberg argues that evolutionary algorithms build up segments of high valued bit strings in the chromosomes of the population (Forrest and Mitchell, 1992). These strings, or building blocks, are built from shortest segments to longer and longer segments. He emphasizes the importance of building block size as a parameter to limit the scope of search for optimization algorithms (Goldberg et al., 1989). Epistasis is a direct measure of the number and maximum size of building blocks.

1.4.2 Linear Functions vs Bitwise Linear Functions

When discussing epistasis it is important to make a clear distinction between fitness functions, f_{fitness}, that are linear functions and those that are bitwise linearly independent functions. A **linear function** is a function $f(x) : \mathcal{B}^L \rightarrow \mathbb{R}$ that can be represented as a polynomial $ax + b$. A **bitwise linear function**, is a function that can be represented as:

$$f(x) = x[0]c_0 + x[1]c_1 + x[2]c_2 + \ldots + x[L-1]c_{l-1} + c$$

where $x[b]$ extracts the b^{th} bit from x and c_i represents the value added to the function's total value if the i^{th} bit is 1. The constant c is an offset that is independent of the value of x. Clearly, a bitwise linearly independent fitness function has no bitwise nonlinearities and hence has at most a single bit of epistasis.

As we shall see, all linear functions are bitwise linearly independent. However, all bitwise linearly independent functions are not linear as we see in this counter example:

x	b_1	b_0	$f(x)$
3	1	1	3
2	1	0	1
1	0	1	2
0	0	0	0

In this case the function cannot be represented as $ax + b$ or even $ax + b \mod c$ and so is not a linear function. However, the function can be written as $b_1 + 2b_0$ and hence is bitwise linearly independent. Notice that encoding is not an issue here, since I am defining these two cases as the full fitness function f_{fitness} which includes encoding.

1.4.3 Other Structure

Other features of fitness functions will be included in this dissertation such as number of local optima, plateaus etc. However, the term structure implies more than the features themselves but rather encompasses the composition of features as well. Informally, **structure** is an *assemblage* by features into a whole. For instance, the distribution and clustering of epistatically interacting bits plays a role in predicting problem difficulty, as we see in a later experiment. Perhaps the most important example is the effect of overlapping epistatically interacting subsets of bits which create constraint satisfaction problems that are similar to k-satisfiability problems (Heckendorn et al., 1999).

1.5 Experiment: Epistasis Related to Difficulty

To further support the hypothesis that epistasis may in some cases be related to problem difficulty I present the results of two experiments. In (Heckendorn et al., 1997) I generated completely random 8 bit functions that had fixed upper limits for the number of bits that could epistatically interact. This upper limit is designated by the Greek letter Ω. For each $\Omega = 1$ through 6, I generated 100 different random functions. An infinite population model for a simple generational genetic algorithm (Vose and Liepins, 1991; Whitley, 1993) was

Table 1.1: Hamming Distance from the Optimum for $\Omega = 1$ through 6.

Ω	Hamming Dist.					% Converged
	0	1	2	3	4	to Nonoptimum
1	100	0	0	0	0	0%
2	78	18	4	0	0	22%
3	78	19	2	1	0	22%
4	60	28	11	1	0	40%
5	56	26	15	2	1	44%
6	52	31	9	7	1	48%

examined after 20 generations. The result is displayed in Table 1.1. This shows for each set of 100 functions. $\Omega = 1$ through 6. the number of times that the function converged to a point at various Hamming distances from the optimum. A Hamming distance of zero. of course. indicates that functions converged to the optimum. Notice that as Ω increases the genetic algorithm increasingly converged to values away from the optimum and that the values were increasingly farther away. This reflects the complexity of the surface created by high levels of bit interaction.

A second experiment was performed in which 30 bit functions were subdivided into contiguous segments of bits Ω long. Each segment was used as the domain to a random function. For example when $\Omega = 3$ the 30 bit domain was subdivided into 10. 3 bit segments. Each segment was assigned random 3 bit function. 100 functions were tested for each Ω. A genetic algorithm was used to solve each to optimality. The results are in Table 1.2. It is clear that for this kind of problem as well that as epistasis increases so does the difficulty as measured by average number of evaluations.

Table 1.2: Average Number of Evaluations to Solve a Segmented Limited Epistasis Problem.

Max Epistasis	Num Evals
1	2600
2	9250
3	30700
4	321000
5	505000
6	558000
7	741000
8	1300000

It is apparent from the *ad hoc* nature of industrial application, and the lack of understanding of the fundamental structure of difficult problems in the research community that there is a need for research relating problem difficulty for evolutionary algorithms and problem structure. It seems clear that epistasis can be an indicator of problem difficulty for many kinds of problems with a random component to them. Unfortunately, we lack the rigor necessary to characterize the epistatic structure of the two classes of problems used in these experiments. In the remaining chapters we will build the mathematical machinery to rigorously analyze the epistatic structure of these kinds of problems and many more.

1.6 What to Expect

This dissertation has a strong theoretical component. Many of the results are in the form of analysis, theorems, notations, and definitions. The scope of this research is constrained to

the relationship between a particular feature of optimization problems known as epistasis and problem difficulty measured by the average performance for evolutionary optimization algorithms in finding a global maximum of the problem. Epistasis will be made precise in the next chapter and further clarified analytically in the chapters to follow. I chose epistasis because it is known to be related to problem difficulty (Davidor, 1991; Reeves and Wright, 1995b), but its influence on problem difficulty is not fully understood. Epistasis is also a fundamental structural property of functions as we will see in later chapters. The remainder of this chapter is devoted to defining many of the terms and notations used the dissertation.

Chapter 2 focuses, in detail, on the basic definitions and theorems of Walsh analysis. Walsh analysis can be used to quantify epistasis via computing Walsh coefficients. Although some of these theorems are well known, such as orthogonality and uniqueness, many are new, such as The Split Mask Theorem and Balanced Sum for Parity. Also, I present a new logic based expression for the Walsh function, the basis of Walsh analysis. Using the logic based notation makes it easier to use logical bit operations on the arguments to the functions. I then discuss the advantages and disadvantages of this expression with respect to other well known expressions for Walsh functions. This chapter provides a firm foundation in Walsh analysis for the rest of the dissertation and is designed to be used as a badly needed reference work on Walsh analysis for members of the evolutionary computation community. Previously, only two very limited works on Walsh analysis were commonly available: the papers by Goldberg (Goldberg, 1989a; Goldberg, 1989b) and Bethke's dissertation (Bethke, 1981).

Since evolutionary algorithms often create new bit strings by slicing and replacing subsets of bits in their populations, it is important to extend epistatic studies to focus on subsets of domain with fixed bit values. One way to represent these subsets of bits is with hyperplanes. If the entire bit space of a function is thought of as a hypercube with elements of the domain at the vertices, a hyperplane is a lower dimensional subcube of the domain delineated by a subset of fixed bits in the domain. If the value of a hyperplane is considered to the be the average of the fitness values of all of the strings in the hyperplane, then hyperplanes seem to compete by this measure, for dominance in the population (Holland,

1975: Whitley et al., 1995a; Heckendorn et al., 1997). Chapter 3 deals with hyperplanes and includes theorems on hyperplane averaging, Pack/Unpack Equivalency, and various function embedding theorems. An application of The Hyperplane Averaging for Numbered Hyperplanes Theorem is given to compute hyperplane averages quickly for all hyperplanes with the same set of fixed bit positions.

Several useful mathematical tools are developed in Chapter 3 as well. The *pack* and *unpack* functions are introduced allowing rigorous definition of extraction of a subset of bits from a larger bit string and creation of a larger bit string out of a smaller one. These functions are used to define hyperplane numbering, which is a compact notation for hyperplanes that will be used to more easily prove theorems later in the dissertation. Spectral functions are also introduced as another compact notation that both aids theorem proving and intuition.

In chapter 3 I also present the idea of function embedding. If function f has an L bit domain and function g has an M bit domain, with $M \leq L$, then f is an embedding of g if f can be defined as g applied to a subset of the bits in the argument to f. Function embedding models a local independent contribution to the overall function value. For instance, an embedded function could represent the contribution to the overall profitability of a factory by values represented in a subset of bits. That subset of bits could be the productivity values of the loading doc, for example. The entire profitability of the factory could then be expressed as a sum of embedded functions. This idea will become the major theme of chapter 6.

Chapters 4 through 6 apply Walsh analysis to various broad classes of problems. I begin chapter 4 by showing how epistasis can be quantified by Walsh analysis and create several more compact measures of epistasis including Walsh counts and sums, and function order. With these as measures of the epistatic structure of a function, I analyze polynomials and show that the maximum number of bits of epistasis of a polynomial is bounded by the degree of the polynomial. Often optimization problems can be considered a composition of decoding functions and an evaluation function. The decoding functions extract and process subsets of bits from the argument bit string and then the results of the decoding

functions are fed to an overall evaluation function. The decoding functions may extract subsets of bits, scale, translate, convert from Gray code, or perform other necessary work. Chapter 4 contains numerous theorems for how this preprocessing of the bit string affects the epistasis as measured by Walsh analysis. I define another property of functions called parity in which functions can have odd and even order. Several theorems on the invariance of these properties are proven. At the end of the chapter, an application of these theorems demonstrates that picking the proper problem representation reduces epistasis, often making the problem easier to solve.

Logical expressions can be used to generate optimization problems either by maximizing a sum of logical expressions or by combining logical expressions as predicates with arithmetic functions. Chapter 5 discusses the Walsh analysis of logical expressions. It is shown that sums of disjunctive and conjunctive clauses creates functions with epistasis bounded by the maximum number of variables in the clauses. Walsh analysis is performed for disjunctive and conjunctive normal clauses. An example is given showing how to use the theorems to perform Walsh analysis of a function that counts the number of one bits in the argument.

In chapter 6 I define a new class of functions called embedded landscapes. These functions are sums of embedded functions and encompass two important classes of functions: NK-landscapes (Kauffman, 1989; Kauffman, 1993) and MAXSAT functions (Papadimitriou, 1994; Eiben and van der Hauw, 1997; Hogg et al., 1996; Mitchell et al., 1992). An analysis of the epistasis of embedded landscapes is performed using Walsh analysis. The results are then applied to both NK-landscapes and MAXSAT problems. I show that all the Walsh coefficients of an embedded landscape, composed of subfunctions with a maximum number of bits in the domain, can be computed in polynomial time. I show how summary statistics (mean, variance, skew, etc.) can also be computed in polynomial time. I conclude that even though all Walsh coefficients of a function can be known in polynomial time, the function can still be NP-complete, e.g. MAX2SAT.

In this dissertation, I have greatly increased the understanding of the epistatic structure of many important problem classes for evolutionary algorithms. I show that MAXSAT problems have low epistasis but are very difficult, in that they are NP-complete. I explain

why epistasis is insufficient to predict problem difficulty and suggest what new problem features we need to consider in order to improve our predictions. I discover that even with complete information about the epistasis of a problem in the form of Walsh coefficients and summary statistics. all computed in polynomial time. the problem may remain NP-complete.

1.7 Notations and Definitions

This is a summary of terms used throughout the dissertation. Most of the ideas will be treated in detail later in the paper. They are brought together here to familiarize the reader with concepts ahead and to act as a reference.

- $\Re$ represents the set of real numbers. Let $\mathbb{Z}_k$ represent the nonnegative integers mod k and $\mathbb{Z}$ represent the nonnegative integers.

- Ranges for summations are sometimes specified by a predicate using a colon. For example: $\sum_{k\,:\,i=k\wedge m}$ reads as the sum over all k such that $i = k \wedge m$. The choices of possible k from which to choose can be inferred from context.

- The modular floor function of n and k is denoted $\lfloor n \rfloor_k$. This returns the largest integer less than or equal to n that is divisible by k. For example $\lfloor 5 \rfloor_3 = 3$ and $\lfloor 6 \rfloor_3 = 6$.

- A **bit space** is built on the set $\mathcal{B} = \{0, 1\}$ and is used to define bit strings. A useful analogous space is the **signed unit space** and is built on the set $\mathcal{Y} = \{1, -1\}$. The function $Y : \mathcal{B} \rightarrow \mathcal{Y}$ maps $0 \rightarrow 1$. $1 \rightarrow -1$. Note that 1 does not map to 1.

- Let a **string** of length L be an ordered list from the set $\mathcal{B}$. The bits in a string are ordered $b_{L-1}, b_{L-2}, \ldots b_2, b_1, b_0$ and belong to $\mathcal{B}^L$. For example: 011001 is from $\mathcal{B}^6$ and has $b_0 = 1$. A string of all 1's will be denoted by $\vec{1}$ and the string of all 0's will be denoted by $\vec{0}$. A string is assumed to be of length L unless otherwise stated.

- For logical operators. I will use $\wedge$ for AND. $\vee$ for OR. $\ominus$ for EXCLUSIVE-OR. and indicate logical NOT by an overline. e.g. $\overline{x}$. If a logical operator is applied to a bit string. it should

be considered to be applied to each bit of the string independently. e.g. $110 \ominus 011 = 101$.

- A function defined over a domain $\mathcal{B}^L$ has a **dimension** L. The dimension of a function f is denoted $dim(f)$.
- Nonnegative integers will be used interchangeably with binary for representing strings in $\mathcal{B}^L$. e.g. $\vec{1}$ may also be referred to as $2^L - 1$. In converting a string to an integer, b_0 is the least significant bit.
- An L bit string can be considered to be a **vector** in L dimensional space. This means that occasionally vector operations can be applied to strings. for example dot product. All vectors will be column vectors unless otherwise specified.
- Let $[i]$ denote extracting the i^{th} bit. So if x =1111011 then $x[2] = 0$. Extracting the bits i through j is denoted $x[i..j]$. For example: $x[2..5] = 1110$.
- $i \subseteq j$ where $i, j \in \mathcal{B}^L$ reads as i **is contained in** j. That is. wherever there is a 1 in i there is a 1 in j or. said another way. $i \wedge \overline{j} = 0$.
- The **bit count** function $bc(i)$ returns the number of 1's in i. For example $bc(001011) = 3$ and $bc(15) = 4$. This is also referred to in the literature as **unitation** (Deb and Goldberg. 1992).
- The **parity function** $parity(i)$ returns 0 if the number of bits in i is even and 1 if the number of bits is odd.
- A **hyperplane** or **schema** is represented by one of the 3^L strings of 0's. 1's and *'s where the 0's and 1's are in the **fixed bit positions** and the *'s represent either a 0 or a 1 in the **variable bit positions**. An example hyperplane h for strings in $\mathcal{B}^7$ might be **1101*. Although both schema and hyperplane are equally valid. I will be using the term hyperplane throughout this paper because I think it is a more mathematically evocative term while schema evokes more of a feeling of process.

- A hyperplane with C fixed bit positions represents a hyperplane of **order** C and defines a set of $2^{(L-C)}$ strings where all possible replacements of the *'s have been defined. For hyperplane h the order of h is denoted by $o(h)$ and the number of strings in h is denoted $|h|$.

- α and β are defined on a hyperplane by the bit by bit mappings below (Goldberg. 1989c):

$$\alpha(h)[i] = \begin{cases} 0 & h[i] = * \\ 1 & h[i] = 0 \text{ or } 1 \end{cases} \qquad \beta(h)[i] = \begin{cases} 0 & h[i] = * \text{ or } 0 \\ 1 & h[i] = 1 \end{cases}$$

  The α returns a mask that identifies the fixed bit positions in the hyperplane. β returns a mask that identifies the bit positions that are set to 1. For the hyperplane $h =$ **1101*: $\alpha(h) =$ 0011110 and $\beta(h) =$ 0011010.

- A **partition** is a set of competing hyperplanes that all share the same set of fixed bit positions. A partition is specified by a string with a b in positions of fixed bit positions and *'s in all other positions. For example, *b** represents a partition that contains the two hyperplanes *1** and *0**.

- A partition with C b's and $(L-C)$ *'s is of **order** C and defines a set of 2^C hyperplanes. For partition π. the order of π is denoted by $o(\pi)$. and the number of hyperplanes in π is denoted $|\pi|$.

- $|x|$ either indicates **absolute value**. when x is a number or **size of** when x is a set such as a hyperplane. partition. or set or numbers.

- $\langle f \rangle_h$ is the average of the values of the function f defined over $\mathcal{B}^L$ for all strings in hyperplane h contained in $\mathcal{B}^L$.

- Functions over a finite domain can also be expressed as a vector of values. I will occasionally express a function $f : \mathcal{B}^L \rightarrow \mathbb{R}$ as a vector in $\mathbb{R}^{(2^L)}$. The order of the elements in the vector will be in ascending order of the binary translation of the bit strings in the domain. The vector representation for function f will be $\vec{f}$.

- The transpose of a vector $\vec{f}$ or matrix M will be denoted with a superscript $\top$. for example. $\vec{f}^{\top}$

1.7.1 $\mathcal{B}$ space and $\mathcal{Y}$ space

I will frequently be dealing with two spaces. One called $\mathcal{B}$ space and the other $\mathcal{Y}$ space. As we have seen. $\mathcal{B}$ **space** is the two element set $\{0.1\}$. $\mathcal{Y}$ **space** is a signed unit analog: $\mathcal{Y} = \{1.-1\}$. $Y : \mathcal{B} \rightarrow \mathcal{Y}$ performs the mapping: $0 \rightarrow 1$. $1 \rightarrow -1$. Note that 1 does not map to 1 in the other space! The Y mapping function will allow us to separate logical operations being done on operands in $\mathcal{B}^K$ space from the numerical operations being done in $\mathbb{R}$. This application can be seen in the next two observations.

Observations: Assume $a.b \in \mathcal{B}$ then:

- $Y(a)Y(b) = Y(a \ominus b)$
- $Y(a)^b = Y(a \wedge b)$

The first of these observations is particularly interesting in that it allows us to move the multiply operator from the world of real number operations inside the Y function where it reappears as a logical operator. As a result. an analogous feature appears in the form of the Walsh Product Theorem in the next chapter.

1.7.2 Representation, Decoding and Neighborhoods

Up until now. I have just considered the simple model of an optimization problem as a single composite fitness function c.f. Equation 1.1 . f_{fitness}. defined over a domain $\mathcal{D}$. There are two common elaborations on this model that should be clarified to better understand the context of the theorems to follow.

The first is an alternate decomposition of f_{fitness}. In this chapter we saw that a fitness function was a composite of three functions. $f_{\text{desirability}}$. f_{model}. and f_{decode}. An alternate decomposition merges $f_{\text{desirability}}$ and f_{model} into a single function. f_{ga} and emphasizes the decoding function:

$$f_{\text{fitness}} = f_{\text{ga}}(f_{\text{decode}}) \tag{1.2}$$

This is a natural separation since it is often the decoding function that the practitioner has the most latitude in changing. The mapping of the argument space of f_{fitness} to the bit space of the chromosome is call an **encoding** and its inverse is represented by the **decoding** function f_{decode}. A model that emphasizes decoding is useful in that the implementor often asks the question. "What is the best encoding for my problem." The answer to this question can radically alter the performance of the resulting algorithm.

The second elaboration on the model of Equation 1.1 is to add structure to the domain space called a neighborhood. A **neighborhood** imposes the idea of adjacency between points in the domain. This adds the concept of adjacency to the domain which makes it possible to think of the problem as a surface, albeit a high dimensional one. From a practical stand point, a neighborhood is often just a convenient way of modeling common paths algorithms use to search through the domain space. Having a neighborhood allows us to define the ideas requiring the concept of "local" such as local optima, basins of attraction, and plateaus. Notice that the distinction between global and local search are based on the notion of locality. Information derived about a function derived from locality may be critical to modeling algorithm performance. On the other hand, a neighborhood may be misleading in that it may not represent the most likely search paths taken by an algorithm. If it does not, the surface created is of little practical predictive value.

Two common encoding functions are the **Gray encoding**, which assumes its argument to be a Gray code for the value to be fed to f_{ga}, and the **binary encoding**, which does the familiar decoding from binary to integers. Notice that if the encoding is Gray then the decoding function, f_{decode} is a degray function.

A **neighborhood** is defined by a set of functions $f_t : \mathcal{B}^L \rightarrow \mathcal{B}^L$ each of which takes a point in the domain and maps it to an "adjacent" point. The neighboring points for a given point p in the domain are the points p_t such that $p_t = f_t(p)$. Two common neighborhoods are the **numerical neighborhood**, which is characterized by the set of two functions $\{(n-1) mod\ 2^L,\ (n+1) \mod 2^L\}$, and the **Hamming neighborhood**, which is characterized by the set of L bit functions $\{(n \oplus 1_2),\ (n \oplus 10_2),\ (n \oplus 100_2),\ \ldots (n \oplus 2^{L-1})\}$.

Often a particular decoding is associated with a particular neighborhood. For example

Hamming neighborhoods are often associated with Gray decodings. A pairing of a neighborhood with a decoding is called a **representation**. Three popular representations are the **Gray representation**. which is a Gray encoding with Hamming neighborhood: the **numeric representation**. which is a binary encoding with numeric neighborhood: and the **binary representation**. which is binary encoding with a Hamming neighborhood.

Technically. a **landscape** is a pairing of a function f_{ga} with a representation. Without a neighborhood structure. expressed or implied. f_{ga} cannot have any local optima or plateaus. A neighborhood may be implied as in the case of NK-Landscape in Kauffman (Kauffman. 1995) which implicitly has a Hamming neighborhood. Unfortunately not all occurrences of NK-Landscape in the literature bother to include the neighborhood definition (Manderick et al.. 1991). In the case of this work I will introduce a class of functions called embedded landscapes. These will technically not be landscapes as their name might imply since they will not have a neighborhood defined for them but rather their name is a historical derivative.

1.8 Summary

The background presented in this chapter defined many of the basic terms that outline my research. In this chapter we saw that there are three important components of my research: problem. algorithm. and difficulty. The problems I want to consider are maximization problems over a domain of bit strings. The algorithms I am considering are evolutionary optimization algorithms that resemble stochastic search. Difficulty is measured in number of fitness function evaluations. I defined epistasis and hope to use it to predict the difficulty of classes of problems relative to evolutionary optimization algorithms. My ultimate goal. expressed in simplistic terms. is:

$$\text{Epistasis(Problem)} \xrightarrow{\text{predicts}} \text{NumEvals(EvolutionaryAlgorithms. Problem)}$$

I support the view that epistasis is related to problem difficulty by giving an argument

by analogy and performing two experiments. However, random functions are involved in both analogy and the experiments. It is possible that, if the functions were not so random, that epistasis might not play as strong a role. At this point it is plain that we lack sufficient mathematical rigor to quantify the epistasis and discuss these structural features. The chapters that follow will lead us to a firm mathematical basis for exploring these ideas.

Chapter 2

The Basics of Walsh Analysis

Walsh analysis (Bethke. 1981) is similar to discrete Fourier analysis. In **discrete Fourier analysis.** a function evaluated at a finite number of points is projected onto an orthogonal set of functions. For Fourier analysis. the functions are sine and cosine functions with varying frequencies. The set of coefficients produced by the projection indicate the frequency content of the function.

Walsh analysis. is similar to discrete Fourier analysis in that it is a projection onto an orthogonal set of functions. The functions in this case are defined over bit strings and return

and the resulting set of coefficients indicate the epistatic content of the function.

In the previous chapter I hypothesized that the larger the number of bits that are epistatically interrelated in a problem. the more difficult the problem should be. In this chapter I develop Walsh analysis as a way of quantifying the epistasis of problems in order to test this hypothesis. Using Walsh analysis for quantifying epistasis is nothing new and was studied both in Bethke's dissertation (Bethke. 1981) and in Goldberg's famous papers on the subject (Goldberg. 1989a). Reeves and Wright strengthened the association between Walsh analysis and epistasis in (Reeves and Wright. 1995b) by the use of experimental design. They show that if an experimental design model is chosed which assumes the value of a function is a constant plus the sum of all linear and nonlinear "fixed effects" between bits then the value of the "fixed effect" differs at most only in sign from the value returned by Walsh analysis for the epistasis. Their results state that. at least in magnitude. that the value Walsh analysis returns as quantifying epistasis between a set of bits is the same as the contributing "effects" of the set bits computed by experimental design. Clearly

understanding Walsh analysis will aid us in quantifying epistasis.

In this chapter I present many basic theorems of Walsh function analysis. The concepts in this chapter may seem too abstract at first but as the chapter proceeds we will be building techniques and intuition about Walsh Functions and Walsh coefficients which will then prove to be useful in studying epistasis. This material will be the foundation for much of the mathematics in the rest of the dissertation. Specifically, I will cover: basic identities, orthogonality theorems, how to efficiently compute Walsh coefficients, vector and matrix representations, and the Walsh transform as a linear transform. All the proofs in this chapter are my proofs. Where the theorems are not new and a reference is known, I will cite it.

2.1 Walsh Polynomials

The Walsh polynomial (seen in Equation 2.1) is the basic transformation from a function defined over bit strings, on the left side of the equation, to a function defined over epistatic interactions on the right side of the equation. It is defined as follows. Any function $f : \mathcal{B}^L \rightarrow \mathbb{R}$ can be broken down into a **Walsh polynomial**:

$$f(x) = \sum_{i=0}^{2^L-1} w_i \psi_i(x) \tag{2.1}$$

where

- $\psi_i(x) : \mathcal{B}^L \rightarrow \{1, -1\}$, $i \in \mathcal{B}^L$, is the i^{th} **Walsh function** of x.
- $w_i \in \mathbb{R}$, $i \in \mathcal{B}^L$, is the i^{th} **Walsh coefficient**. w_i indicates the degree of interaction of bits indicated by the positions of the 1's in i. The **order of Walsh coefficient**, $|w_i|$, is the number of one bits in i.
- w_i is defined to be the coefficient associated with the i^{th} Walsh function $\psi_i(x)$. In proofs throughout this paper, the value of w_i is defined as the coefficient associated with Walsh function $\psi_i(x)$.

The most important feature of a Walsh polynomial is that any function f can be uniquely

decomposed into a linear weighted sum of Walsh functions where the weights are the real valued Walsh coefficients.

This equation defines a mapping from the 2^L values in the table that defines the Walsh coefficients to another table of 2^L values that defines the function f. Note that **both** i and x are elements of $\mathcal{B}^L$. $f(x)$ and w_i may be real numbers, but x is not. This is often a point of confusion. It is also important to note that the Walsh coefficients of the sum of two functions is the sum of the Walsh coefficients of the two functions.

Walsh functions are defined as functions, $\psi_i(x)$, that return a value based on the parity of the AND of x and i. An even number of 1 bits returns a 1, otherwise it returns a -1. This can be expressed in logic notation as:

$$\psi_j(x) = Y(\bigoplus_{i=0}^{L-1}(x[i] \wedge j[i])) \qquad x, j \in \mathcal{B}^L$$

where the $\oplus$ in this case denotes an iterated EXCLUSIVE-ORover the following expression and $x[i]$ indicates the i^{th} bit in bit string x. For example: $\psi_{01110}(11010)$ can be computed by first taking the bitwise AND of 01110 and 11010 yielding 01010. The parity of this string is computed using EXCLUSIVE-OR giving 0. The Y function, defined in the notation section, maps this to $+1$. The function ψ is often written with a nonbinary subscript. In this example, ψ_{01110} would be written ψ_{14} which would be referred to as the 14^{th} Walsh function.

Although there are many ways to define this function, this way compartmentalizes the logical expression nicely from the real-valued numerical portion via the interface function Y. The Y function, as described in the notation section, maps the logical values $\{0, 1\}$ to the numerical values $\{1, -1\}$. The Y function could be replaced by raising -1 to the argument of Y but this would lead to a lot of the mathematics in a proof appearing in the cramped exponent portion of the expressions.

The three most important intuitive properties to remember about Walsh functions are:

- **Symmetry** - The values of x and j can be interchanged without effecting the value of the function, i.e. $\psi_i(j) = \psi_j(i)$.

- **Parity** - The exclusive-or in the Walsh Function computes the parity of the results of the AND's.

- **Masking** - The value of j (or x) can be thought of as a mask to select the bits in x (or j) for which we are interested in taking the parity measure.

Why are Walsh functions a good idea?

- Walsh function notation for a function is a *linear* function of Walsh coefficients and Walsh functions and therefore easy to manipulate and use.

- Walsh functions provide a uniform way of treating bit interactions.

- Walsh functions make it easy to calculate effects of any L bit combination on the value of the function over its whole range.

- Walsh function notation provides a bit interaction representation for a function.

- The + and − signs of Walsh functions create opportunities for cancellation of terms of opposite sign.

- The set of Walsh functions for a given bit space form an orthogonal set of functions.

2.1.1 Alternate Notations

Many different definitions for Walsh functions can be found in the literature. Each can be justified as being for a different purpose or to emphasize a particular aspect of the function.

In (Goldberg, 1989a), Walsh functions are defined using a product/power form:

$$\psi_j(x) = \prod_{i=0}^{L-1} Y^{(x[i])^{j[i]}}$$

where $[i]$ denotes extracting the i^{th} bit. Soraya Rana has suggested:

$$\psi_j(x) = -1^{bc(x \wedge j)}$$

I have proposed the various other parity based expressions for a Walsh function:

$$\psi_j(x) = -1^{parity(x \wedge j)} = 1 - 2\,parity(x \wedge j)$$

where $parity(x)$ is 0 if $bc(x)$ is even, and 1 if $bc(x)$ is odd. If j and x are considered column vectors of bits, then ψ can be written:

$$\psi_j(x) = -1^{x^T j}$$

where $x^T j$ is just the dot product of the bit vectors. Similarly, Manela and Campbell (Manela and Campbell, 1992) use the nearly identical expression:

$$\psi_j(x) = (-1)^{\sum_{i=0}^{L-1} x[i]j[i]}$$

The field of Harmonic Analysis approaches the same problem from the point of view of n-dimensional discrete Fourier analysis (Lechner, 1971).

$$\psi_j(x) = \exp(2\pi i(x^T j)/2)$$

2.2 Basic Walsh Function Theorems

The elementary theorems of Walsh functions tell us something about the character of Walsh functions. The Identity Theorem states that regardless of which Walsh function you choose, if the argument is zero the result is 1. This property is useful in cancellation of subexpressions in later proofs. This theorem is well known and I believe is assumed too trivial to prove. The theorem can be inferred from Figure 1 in (Goldberg, 1989a).

Theorem 1 (Identity Theorem)

$$\psi_j(0) = 1 \qquad \forall j$$

Proof:

$$\begin{aligned}
\psi_j(0) &= Y^{\cdot}(\bigoplus_{i=0}^{L-1}(0[i] \wedge j[i])) \\
&= Y^{\cdot}(\bigoplus_{i=0}^{L-1}(0 \wedge j[i])) \\
&= Y^{\cdot}(\bigoplus_{i=0}^{L-1} 0) \\
&= Y^{\cdot}(0) \\
&= 1
\end{aligned}$$

□

The common notation for Walsh functions gives the inaccurate impression that somehow function number and argument are very different. The Symmetry Theorem proves they aren't at all and provides us with a very useful transformation.

Theorem 2 (Symmetry)

$$\psi_j(k) = \psi_k(j)$$

Proof:

$$\begin{aligned}
\psi_j(k) &= Y^{\cdot}(\bigoplus_{i=0}^{L-1}(k[i] \wedge j[i])) \\
&= Y^{\cdot}(\bigoplus_{i=0}^{L-1}(j[i] \wedge k[i])) \\
&= \psi_k(j)
\end{aligned}$$

□

This next theorem shows us that not only is ψ_0 a constant function but provides us our first use of Symmetry.

Theorem 3 (Constant Walsh Function)

$$\psi_0(j) = 1 \qquad \forall j$$

Proof:

Obvious from theorems 1 and 2.

□

The next theorem is a precursor to several useful corollaries and is simply based on the fact that the arguments to ψ are first ANDed together.

Theorem 4 (Split Mask)

$$\psi_{j \wedge m}(k) = \psi_{j \wedge a}(k \wedge b) \qquad \text{where} \quad a \wedge b = m$$

Proof:

$$\begin{aligned}
\psi_{j \wedge m}(k) &= Y(\bigoplus_{i=0}^{L-1}(\ (j[i] \wedge m[i]) \ \wedge k[i])) \\
&= Y(\bigoplus_{i=0}^{L-1}(\ (j[i] \wedge (a[i] \wedge b[i])) \ \wedge k[i])) \\
&= Y(\bigoplus_{i=0}^{L-1}((j[i] \wedge a[i]) \ \wedge \ (b[i]) \wedge k[i])) \\
&= \psi_{j \wedge a}(k \wedge b)
\end{aligned}$$

□

The following corollaries are particularly useful special forms of the Split Mask Theorem. Conjunctive Compression follows directly from the Split Mask Theorem. It allows us to limit the range of Walsh functions used by the range of arguments used.

Corollary 4a (Conjunctive Compression)

$$\psi_j(k) = \psi_{j \wedge k}(k)$$

□

Associative Masking allows us to apply a mask to either the function index or the function argument.

Corollary 4b (Associative Masking)

$$\psi_{j \wedge m}(k) = \psi_j(k \wedge m)$$

□

Similar to the Conjunctive Compression Corollary is the Disjunctive Compression Theorem.

Theorem 5 (Disjunctive Compression)

$$\psi_k(k) = \psi_{j \vee k}(k)$$

Proof:

$$\begin{aligned} \psi_{j \vee k}(k) &= \Upsilon(\bigoplus_{i=0}^{L-1}((j[i] \vee k[i]) \wedge k[i])) \\ &= \Upsilon(\bigoplus_{i=0}^{L-1}((k[i] \wedge k[i])) \\ &= \psi_k(k) \end{aligned}$$

□

By setting $j = \vec{1}$ in the above theorem, we get the following useful corollary which allows us, in special cases, to convert one of the function index to a constant.

Corollary 5a

$$\psi_k(k) = \psi_{\vec{1}}(k)$$

□

We can use this immediately to relate Walsh functions and parity.

Theorem 6 (Walsh Parity Relation)

$$parity(k) = (1 - \psi_k(k))/2$$

Proof:

One of the alternative expressions for a Walsh function was:

$$\psi_j(k) = 1 - 2\,parity(k \wedge j)$$

Setting $j = \vec{1}$ gives us:

$$\begin{aligned} \psi_{\vec{1}}(k) &= 1 - 2\,parity(k) \\ \psi_k(k) &= 1 - 2\,parity(k) \\ (1 - \psi_k(k))/2 &= parity(k) \end{aligned}$$

□

The next theorem is one of the most useful of the basic Walsh function theorems. It allows us to collapse the product of two Walsh functions into a single one. It also allows one to change a logical operator that is an argument to ψ to a numerical operator which is multiplication. Because the theorem is so useful, it is ubiquitous in this dissertation. Therefore, it is wise for the reader to be sure they understand it clearly before proceeding.

The proof of the theorem takes advantage of the fact that ψ is defined with an EXCLUSIVE-OR as the outer most operator.

Theorem 7 (Walsh Product)

$$\psi_j(x)\psi_k(x) = \psi_{j \dot{+} k}(x)$$

Proof:

$$\begin{aligned}
\psi_j(x)\psi_k(x) &= Y(\bigoplus_{i=0}^{L-1}(j[i] \wedge x[i]))\ \ Y(\bigoplus_{i=0}^{L-1}(k[i] \wedge x[i])) \\
&= Y(\ (\bigoplus_{i=0}^{L-1}(j[i] \wedge x[i])) \ominus (\bigoplus_{i=0}^{L-1}(k[i] \wedge x[i]))\ ) \\
&= Y(\ \bigoplus_{i=0}^{L-1}((j[i] \wedge x[i]) \ominus (k[i] \wedge x[i]))\ ) \\
&= Y(\ \bigoplus_{i=0}^{L-1}((j[i] \wedge k[i]) \ominus x[i])\ ) \\
&= \psi_{j \doteq k}(x)
\end{aligned}$$

□

To drive home the symmetry property I provide this theorem as an example of its use.

Corollary 7a (A Use of the Walsh Product Theorem)

$$\psi_x(j)\psi_x(k) = \psi_x(j \ominus k)$$

Proof:

The proof follows directly by use of the Product and Symmetry Theorems.

□

The Negation Theorem shows that the Walsh function of the negation of a number is related to the Walsh function of the number by the parity of the Walsh function index. The proof of the Negation Theorem uses $2^L - 1$ to give a string of all 1's.

Theorem 8 (Negation)

$$\psi_j(\ \overline{k}\ ) = (-1)^{bc(j)}\psi_j(k)$$

Proof:

$$\begin{aligned}\psi_j(\overline{k}) &= \psi_j((2^L-1)\ominus k)\\ &= \psi_j((2^L-1))\psi_j(k)\\ &= (-1)^{bc(j)}\psi_j(k)\end{aligned}$$

□

The Balanced Sum Theorem is one of two theorems critical in cancelling out sums of Walsh functions. Also, the proof itself is important as a prototype for several proofs to come where equal sets of strings of opposite sign cancel. We will see this form of proof again.

Theorem 9 (Balanced Sum)

$$\sum_{x=0}^{2^L-1}\psi_j(x) = \begin{cases} 2^L & \text{if } j=0\\ 0 & \text{if } j\neq 0\end{cases}$$

Proof:

First observe that this is a problem of counting $+1$'s and -1's

CASE 1: if $j=0$ then

$$\begin{aligned}\textstyle\sum_{x=0}^{2^L-1}\psi_j(x) &= \textstyle\sum_{x=0}^{2^L-1}\psi_0(x)\\ &= \textstyle\sum_{x=0}^{2^L-1}1\\ &= 2^L-1\end{aligned}$$

CASE 2: if $j\neq 0$ then $\exists k$ such that $j[k]=1$. We now divide the 2^L x values into 2 sets.

Let A be the set of all $\psi_j(x)$ where $x[k]=1$.

Let B be the set of all $\psi_j(x)$ where $x[k]=0$.

If $x \in A$. then $x \ominus 2^k \in B$. Let $y = x \ominus 2^k$. Note that for every $x \in A$ there is a unique $y \in B$ and vice versa: therefore $|A| = |B| = 2^{L-1}$.

$$\psi_j(y) = Y(\bigoplus_{i=0}^{L-1}(y[i] \wedge j[i]))$$

Using x. y and k as above. we know that $y[k] = x[k] \ominus 1$. So. for the k^{th} term in the above EXCLUSIVE-OR:

$$\begin{aligned}(y[k] \wedge j[k]) &= ((x[k] \ominus 1) \wedge j[k]) \\ &= (x[k] \wedge j[k]) \ominus 1)\end{aligned}$$

Bringing the 1 out of the EXCLUSIVE-OR:

$$\begin{aligned}\psi_j(x) &= Y(1 \ominus (\bigoplus_{i=0}^{L-1}(x[i] \wedge j[i]))) \\ &= Y(1)Y(\bigoplus_{i=0}^{L-1}(x[i] \wedge j[i]))) \\ &= -1\psi_j(y)\end{aligned}$$

Since there is a one to one onto correspondence between elements of A and elements of B and each pair is of opposite sign. the sum is zero.

□

The Orthogonality Theorem is the second critical theorem for cancellation of sums of Walsh functions. The theorem and a brief outline of the proof appears in (Goldberg. 1989a). The term orthogonal comes from the similar concept of orthogonal functions where the sum in the theorem would be replaced by an integral over a range. A second interpretation as vectors will be introduced in the section on Walsh matrices.

Theorem 10 (Orthogonality)

$$\sum_{x=0}^{2^L-1} \psi_i(x)\psi_j(x) = \begin{cases} 2^L & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Proof:

$$\sum_{x=0}^{2^L-1} \psi_i(x)\psi_j(x) = \sum_{x=0}^{2^L-1} \psi_{i \ominus j}(x)$$

If $i = j$ then $i \ominus j = 0$ and if $i \neq j$ then $i \ominus j \neq 0$. Therefore, by The Balanced Sum Theorem, the theorem is proved.

□

An alternative form of the orthogonality theorem is:

$$\sum_{x=0}^{2^L-1} \psi_i(x)\psi_x(j) = \begin{cases} 2^L & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

This follows immediately by the Symmetry Theorem. This form is similar to the operation in a matrix multiply as we shall soon see.

A law that is similar to the Balanced Sum theorem exists for several subsets of strings in the bit space.

Theorem 11 (Balanced Sum for Parity)

$$\sum_{x:parity(x)=1} \psi_j(x) = \begin{cases} 2^{L-1} & \text{if } j = 0 \\ -2^{L-1} & \text{if } j = 2^L - 1 \\ 0 & \text{otherwise} \end{cases}$$

and

$$\sum_{x:parity(x)=0} \psi_j(x) = \begin{cases} 2^{L-1} & \text{if } j = 0 \\ 2^{L-1} & \text{if } j = 2^L - 1 \\ 0 & \text{otherwise} \end{cases}$$

Proof:

$$parity(k) = (1 - \psi_k(k))/2$$

$$\begin{aligned}
\textstyle\sum_{x:parity(x)=1} \psi_j(x) &= \textstyle\sum_{x=0}^{2^L-1} parity(x)\psi_j(x) \\
&= \textstyle\sum_{x=0}^{2^L-1} \frac{1}{2}(1 - \psi_x(x))\psi_j(x) \\
&= \tfrac{1}{2}(\textstyle\sum_{x=0}^{2^L-1} \psi_j(x) \quad - \quad \sum_{x=0}^{2^L-1} \psi_x(x)\psi_j(x)) \\
&= \tfrac{1}{2}(\textstyle\sum_{x=0}^{2^L-1} \psi_j(x) \quad - \quad \sum_{x=0}^{2^L-1} \psi_{\bar{1}}(x)\psi_j(x)) \\
&= \tfrac{1}{2}(\textstyle\sum_{x=0}^{2^L-1} \psi_j(x) \quad - \quad \sum_{x=0}^{2^L-1} \psi_{\bar{1}+j}(x)) \\
&= \tfrac{1}{2}(\textstyle\sum_{x=0}^{2^L-1} \psi_j(x) \quad - \quad \sum_{x=0}^{2^L-1} \psi_{\bar{j}}(x))
\end{aligned}$$

Both terms can be reduced by the Balanced Sum Theorem

$$= \begin{cases} 2^{L-1} & \text{if } j = 0 \\ -2^{L-1} & \text{if } j = 2^L - 1 \\ 0 & \text{otherwise} \end{cases}$$

therefore the first part is proven. The second part is now easy by observing that:

$$\sum_{x=0}^{2^L-1} \psi_j(x) = \sum_{x:parity(x)=1} \psi_j(x) + \sum_{x:parity(x)=0} \psi_j(x)$$

This observation about parity will recur in discussions of parity in later chapters.

2.3 Computing Walsh Coefficients

It is possible to compute the values of the Walsh coefficients given all the values of a function. This theorem was stated in (Goldberg. 1989a) but never proven.

Theorem 12 (Computing Walsh Coefficients)

$$w_j = \frac{1}{2^L} \sum_{x=0}^{2^L-1} f(x)\psi_j(x) \tag{2.2}$$

Proof:

$$\begin{aligned}
\textstyle\sum_{x=0}^{2^L-1} f(x)\psi_j(x) &= \textstyle\sum_{x=0}^{2^L-1} \psi_j(x)(\sum_{z=0}^{2^L-1} w_z\psi_z(x)) \\
&= \textstyle\sum_{z=0}^{2^L-1} w_z \sum_{x=0}^{2^L-1} \psi_j(x)\psi_z(x) \\
&= \textstyle\sum_{z=j}^{j} w_z 2^L \\
&= w_j 2^L \\
\tfrac{1}{2^L}\textstyle\sum_{x=0}^{2^L-1} f(x)\psi_j(x) &= w_j
\end{aligned}$$

From this it is plain that w_0 is the average of all of the function values and hence plays the role of a constant offset in the expression of a function as a Walsh polynomial.

Theorem 13 (Mean Value)

w_0 = the average value of $f(x)$ over its entire domain.

Proof:

$$w_0 = \frac{1}{2^L} \sum_{x=0}^{2^L-1} f(x)\psi_0(x)$$

but $\psi_0(x) = 1$ therefore

$$w_0 = \frac{1}{2^L} \sum_{x=0}^{2^L-1} f(x)$$

□

The Function Negation theorem follows as a direct and obvious consequence of the definition of the Walsh coefficients.

Theorem 14 (Function Negation)

If function f has Walsh coefficients w_i^f then the function $-f$ has Walsh coefficients $-w_i^f$.

Proof:

Trivially:

$$w_j = \frac{1}{2^L} \sum_{x=0}^{2^L-1} -f(x)\psi_j(x) = -\frac{1}{2^L} \sum_{x=0}^{2^L-1} f(x)\psi_j(x) = -w_j$$

□

The **reflection** of f about the **reflection point** z can be defined as $g(x) = f(x \ominus z)$. It is not obvious that regardless of about what point in $\mathcal{B}^L$ we reflect the arguments to a function, only the *signs* of the resulting Walsh coefficients change. This next theorem provides the proof and will be useful in discussing function parity later.

Theorem 15 (Reflection of Function Arguments)

Let function $f(x)$ have Walsh coefficients w_i^f and $g(x) = f(x \ominus z)$. Then $w_i^g = \psi_i(z) w_i^f$.

Proof:

$$
\begin{aligned}
w_j^g = & \ \frac{1}{2^L} \sum_{x=0}^{2^L-1} f(x \ominus z) w_j(x) \\
& \text{letting } x' = x \ominus z \\
& \frac{1}{2^L} \sum_{x'=0}^{2^L-1} f(x') w_j(x' \ominus z) \\
& \frac{1}{2^L} \sum_{x'=0}^{2^L-1} f(x') w_j(x') w_j(z) \\
& w_j(z) \frac{1}{2^L} \sum_{x'=0}^{2^L-1} f(x') w_j(x') \\
& w_j(z) w_j^f
\end{aligned}
$$

□

It is clear in the previous proof that the Walsh coefficients of g are off by at most their sign from those of f. But does each z produce a unique set of Walsh coefficients? It must be true if the Walsh functions, w_z, are unique.

Theorem 16 (Uniqueness of Walsh Functions)

For any y and z, $y \neq z$, $\exists j$ such that $w_j(y) \neq w_j(z)$.

Proof:

If $y \neq z$ then $y = z \ominus (z \ominus y)$ where $z \ominus y \neq 0$. Therefore

$$
\begin{aligned}
w_j(y) & = w_j(z \ominus (z \ominus y)) \\
& = w_j(z) w_j(z \ominus y)
\end{aligned}
$$

Now select j such that $j \subseteq (z \ominus y)$ and $bc(j) = 1$. By the definition of a Walsh function $w_j(z \ominus y) = -1$. Hence

$$
w_j(y) = -w_j(z)
$$

and the theorem is proven.

□

This theorem will be a natural consequence of the fact that Walsh matrices, which we will develop later in the chapter, will turn out to be invertible. This means that there are 2^L reflections for an L bit function and 2^L different assignments of sign for the Walsh coefficients for reflected functions.

2.4 Products, Convolutions and the Transform Function

Since the Walsh polynomial is a weighted sum of Walsh functions, it is clear the Walsh coefficients of the sum of two functions is the sum of the Walsh coefficients. More precisely:

Theorem 17 (Sum Coefficient)

Let w^f be the Walsh coefficients of f and w^g the Walsh coefficients of g. Then the Walsh coefficients of the sum $f+g$, denoted by w^{fg}, are:

$$w_k^{fg} = w_k^f + w_k^g$$

□

The next theorem shows that the product and convolution are related much as they are in discrete Fourier analysis (Weaver, 1989). Here, convolution is defined using EXCLUSIVE-OR and so the extension of a function to be periodic, as in discrete Fourier analysis, is unnecessary.

Theorem 18 (Product Coefficient)

Let w^f be the Walsh coefficients of f and w^g the Walsh coefficients of g. Then the Walsh coefficients of the product fg, denoted by w^{fg}, are:

$$w_k^{fg} = \sum_{j=0}^{2^L-1} w_j^f w_{k \div j}^g$$

Proof:

Let f and g be defined by their Walsh expansions as before.

$$
\begin{aligned}
f(x)g(x) &= (\textstyle\sum_{j=0}^{2^L-1} w_j^f \psi_j(x))(\sum_{k=0}^{2^L-1} w_k^g \psi_k(x)) \\
&= \textstyle\sum_{j=0}^{2^L-1} \sum_{k=0}^{2^L-1} w_j^f w_k^g \psi_k(x)\psi_j(x) \\
&= \textstyle\sum_{j=0}^{2^L-1} \sum_{k=0}^{2^L-1} w_j^f w_k^g \psi_{k \ominus j}(x) \\
&\quad \text{let } k' = j \ominus k, \text{ therefore } k = k' \ominus j \\
&= \textstyle\sum_{j=0}^{2^L-1} \sum_{k'=0}^{2^L-1} w_j^f w_{k' \ominus j}^g \psi_{k' \ominus j \ominus j}(x) \\
&= \textstyle\sum_{j=0}^{2^L-1} \sum_{k'=0}^{2^L-1} w_j^f w_{k' \ominus j}^g \psi_{k'}(x) \\
&= \textstyle\sum_{k'=0}^{2^L-1} \left(\sum_{j=0}^{2^L-1} w_j^f w_{k' \ominus j}^g \right) \psi_{k'}(x)
\end{aligned}
$$

Therefore the Walsh coefficient for the product of f and g is:

$$
w_{k'}^{fg} = \sum_{j=0}^{2^L-1} w_j^f w_{k' \ominus j}^g
$$

□

This theorem introduces some interesting relations between the convolution of two functions and their product. The **convolution of an infinite sequence** is generally computed (Graham et al., 1989):

$$
c_n = \sum_{k=0}^{n} a_k b_{n-k} \quad \text{for} \quad n \in \{0, 1, 2, \ldots\}
$$

that is, all pairs of elements, one from a and one from b, are summed such that the subscripts sum to a constant n. In general, c_n is formed by the sum of $n + 1$ pairs.

For our purposes, we will define the **convolution of two L bit functions** a and b as:

$$
c(n) = \sum_{k=0}^{2^L-1} a(k)b(n \ominus k) \quad \text{for} \quad n \in \{0, 1, 2, \ldots, 2^L - 1\}
$$

In this form of convolution, all pairs of elements, one from a and one from b, are summed where the EXCLUSIVE-OR of the subscripts themselves results in a constant n. In this case, $c(n)$ is formed by the sum of 2^L pairs regardless of the value of n. The convolution of two functions, f and g, is denoted $f \circ g$.

The Product Coefficient Theorem can now be interpreted in terms of a convolution as follows: the Walsh coefficient of the product of two functions is equal to the convolution of the Walsh coefficients of the separate functions. A further discussion of this interpretation can be found in Lechner (Lechner, 1971) and in Manela and Campbell (Manela and Campbell, 1992).

2.5 Duality and the Walsh Transform

The obvious similarity between the formula for the function, f,

$$f(x) = \sum_{i=0}^{2^L-1} w_i \psi_i(x)$$

and the formula for computing Walsh coefficients, w,

$$w_j = \frac{1}{2^L} \sum_{x=0}^{2^L-1} f(x)\psi_j(x)$$

leads one to consider the duality of the functions and coefficients. Since the function f is over a finite discrete domain, both f and w can be treated as tables of values. In fact, it might make more notational sense to express $f(x)$ as f_x to show the symmetry between the two tables. Also, since $\psi_i(x)$ is more like a commutative operator it might make more sense to express $\psi_i(x)$ as $\psi(i,x)$ to show the commutative nature of the function. In fact, many of the early proofs about Walsh functions follow immediately from theorems of basic symbolic logic if you consider, that for the most part, the binary operator $\psi(i,x)$ can be thought of as a logical AND performed in parallel.

To emphasize the symmetry I express these two formulas in a more symmetric notation:

$$w_j = \frac{1}{2^L} \sum_{i=0}^{2^L-1} f_i \psi(i,j)$$

$$f_j = \sum_{i=0}^{2^L-1} w_i \psi(i,j)$$

The first equation is known as the **Walsh transform** and the second as the **inverse Walsh transform**. The summations in the two formulas are essentially the same and can be computed with the same software. In fact, the two transforms could be made to be identical by multiplying the expression for f_j by a scaling factor of $\frac{1}{\sqrt{2^L}}$ and dividing the expression for w_j by the same factor. I chose to avoid the introduction of the messy irrational values into the equations. It is purely a personal preference and does not effect the results.

This notational example is used to illustrate that Walsh space and function space are duals of each other much as space and time are duals in Fourier analysis.

2.6 Walsh Matrices

In the previous section we mentioned how both the function f_j and the Walsh coefficients w_j could be thought of as finite length tables. In fact, it is useful, to think of the function table as a **function vector** and the Walsh coefficients as a **Walsh coefficient vector**. It may then be possible to operate on them with a matrix of Walsh functions. In this section, we show how this is done and develop some related theorems.

We begin by looking at an example. The following equations show, in excruciating detail, how a function $f : \mathcal{B}^3 \rightarrow \Re$ would be computed using the inverse Walsh transform in the previous section.

$$
\begin{aligned}
f(0) &= \psi_0(0)w_0 + \psi_1(0)w_1 + \psi_2(0)w_2 + \psi_3(0)w_3 + \psi_4(0)w_4 + \psi_5(0)w_5 + \psi_6(0)w_6 + \psi_7(0)w_7 \\
f(1) &= \psi_0(1)w_0 + \psi_1(1)w_1 + \psi_2(1)w_2 + \psi_3(1)w_3 + \psi_4(1)w_4 + \psi_5(1)w_5 + \psi_6(1)w_6 + \psi_7(1)w_7 \\
f(2) &= \psi_0(2)w_0 + \psi_1(2)w_1 + \psi_2(2)w_2 + \psi_3(2)w_3 + \psi_4(2)w_4 + \psi_5(2)w_5 + \psi_6(2)w_6 + \psi_7(2)w_7 \\
f(3) &= \psi_0(3)w_0 + \psi_1(3)w_1 + \psi_2(3)w_2 + \psi_3(3)w_3 + \psi_4(3)w_4 + \psi_5(3)w_5 + \psi_6(3)w_6 + \psi_7(3)w_7 \\
f(4) &= \psi_0(4)w_0 + \psi_1(4)w_1 + \psi_2(4)w_2 + \psi_3(4)w_3 + \psi_4(4)w_4 + \psi_5(4)w_5 + \psi_6(4)w_6 + \psi_7(4)w_7 \\
f(5) &= \psi_0(5)w_0 + \psi_1(5)w_1 + \psi_2(5)w_2 + \psi_3(5)w_3 + \psi_4(5)w_4 + \psi_5(5)w_5 + \psi_6(5)w_6 + \psi_7(5)w_7 \\
f(6) &= \psi_0(6)w_0 + \psi_1(6)w_1 + \psi_2(6)w_2 + \psi_3(6)w_3 + \psi_4(6)w_4 + \psi_5(6)w_5 + \psi_6(6)w_6 + \psi_7(6)w_7 \\
f(7) &= \psi_0(7)w_0 + \psi_1(7)w_1 + \psi_2(7)w_2 + \psi_3(7)w_3 + \psi_4(7)w_4 + \psi_5(7)w_5 + \psi_6(7)w_6 + \psi_7(7)w_7
\end{aligned}
$$

Evaluating the Walsh functions, we get:

$$
\begin{aligned}
f(0) &= +w_0 + w_1 + w_2 + w_3 + w_4 + w_5 + w_6 + w_7 \\
f(1) &= +w_0 - w_1 + w_2 - w_3 + w_4 - w_5 + w_6 - w_7 \\
f(2) &= +w_0 + w_1 - w_2 - w_3 + w_4 + w_5 - w_6 - w_7 \\
f(3) &= +w_0 - w_1 - w_2 + w_3 + w_4 - w_5 - w_6 + w_7 \\
f(4) &= +w_0 + w_1 + w_2 + w_3 - w_4 - w_5 - w_6 - w_7 \\
f(5) &= +w_0 - w_1 + w_2 - w_3 - w_4 + w_5 - w_6 + w_7 \\
f(6) &= +w_0 + w_1 - w_2 - w_3 - w_4 - w_5 + w_6 + w_7 \\
f(7) &= +w_0 - w_1 - w_2 + w_3 - w_4 + w_5 + w_6 - w_7
\end{aligned}
$$

If we consider that the function values and Walsh coefficients can be written as column vectors from $\Re^8$, then we can compute the function vector as the product of the Walsh

coefficient vector and a matrix of Walsh functions. Let the matrix of functions $\psi_c(r)$ define a **Walsh matrix**, Ψ, where r is the row index and c is the column index. Specifically, Ψ for an L bit function is a $2^L \times 2^L$ matrix denoted Ψ_L. Note that the value of Ψ_L is only dependent on the function size L, not on the particular $\vec{w}$ or $\vec{f}$.

The matrix that follows is for Ψ_3. Lines have been added to make it easier to see the symmetry in the various quadrants.

$$\Psi_3 = \left[\begin{array}{cc|cc|cccc} +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 \\ \hline +1 & +1 & -1 & -1 & +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 & +1 & -1 & -1 & +1 \\ \hline +1 & -1 & +1 & +1 & -1 & -1 & -1 & -1 \\ +1 & -1 & +1 & -1 & -1 & +1 & -1 & +1 \\ +1 & +1 & -1 & -1 & -1 & -1 & +1 & +1 \\ +1 & -1 & -1 & +1 & -1 & +1 & +1 & -1 \end{array}\right]$$

Therefore, if we let the column vector $\vec{f}$ represent the function and column vector $\vec{w}$ represent the Walsh coefficients, then it is clear that:

$$\Psi \vec{w} = \vec{f}$$

It is also clear that the matrix is symmetric

$$\Psi = \Psi^{\mathsf{T}}$$

which is what we would expect from the symmetry theorem which stated $\psi_j(k) = \psi_k(j)$.

2.6.1 Uniqueness

Note that any Ψ_L matrix is symmetric and can be arranged as a block matrix:

$$\Psi_L = \begin{bmatrix} A & A \\ A & -A \end{bmatrix}$$

where A is Ψ_{L-1}. That is

$$\Psi_L = \begin{bmatrix} \Psi_{L-1} & \Psi_{L-1} \\ \Psi_{L-1} & -\Psi_{L-1} \end{bmatrix}$$

The initial case is:

$$\Psi_1 = \begin{bmatrix} 1 \end{bmatrix}$$

Using the block matrix form I can determine a recursive formula for the determinant of the matrix.

Theorem 19 (Ψ's Determinant)

$$\det(\Psi_L) = (-2)^{(L2^{L-1})}$$

Proof:

The proof proceeds by developing an upper triangular matrix. The determinant will then be the product of the diagonal elements.

For $L = 1$ the determinant is -2. by inspection.

For $L > 1$ the upper half of the Ψ_L matrix is subtracted from the lower half giving a new matrix with the same determinant as the original:

$$\Psi_3' = \begin{bmatrix} A & A \\ 0 & -2A \end{bmatrix}$$

where $\det(\Psi_3') = \det(\Psi_3)$.

This means that **each** eventual diagonal element in the lower right quadrant will be multiplied by -2. If D_L is the determinant of the $2^L \times 2^L$ matrix. Ψ'. D_L can be expressed recursively as:

$$D_L = \begin{cases} -2 & \text{if } L = 1 \\ (-2)^{2^{L-1}} D_{L-1}^2 & \text{if } L > 1 \end{cases}$$

In the case $L > 1$. one D_{L-1} in the square term comes from the determinant of the upper left block in the matrix. The power of 2^{L-1} comes from the 2^{L-1} doublings of the lower right quadrant and the remaining D_{L-1} in the square term comes from the original determinant of the lower right quadrant. The recursive relation is solved in log space.

Let $d_L = \log_2 D_L$. Then the recurrence relation for $L > 1$ becomes

$$\begin{aligned} D_L &= 2^{2^{L-1}} D_{L-1}^2 \\ \log_2 D_L &= 2^{L-1} + 2\log_2 D_{L-1} \\ d_L &= 2^{L-1} + 2d_{L-1} \end{aligned}$$

Solving for d_L gives: $d_L = L2^{L-1}$. Therefore $D_L = (-2)^{L2^{L-1}}$. The D_L for $L = 1$ is consistent with the formula for $L > 1$ giving a result valid for all L.

Note that for $L > 1$. all exponents $L2^{L-1}$ are even: so the determinant is positive. However. what is most important here is the determinant is nonzero. giving a **unique**

inverse for each Ψ_L.

Theorem 20 (Unique Mapping)

The mapping $\vec{f} \rightarrow \vec{w}$ is unique and reversible.

Proof:

Since the determinant of Ψ is nonzero there exists a unique Ψ^{-1} such that:

$$\vec{w} = \Psi^{-1}\vec{f}$$

□

It is easy to determine this unique inverse.

Theorem 21 (Ψ's Inverse)

$$\Psi_L^{-1} = \frac{1}{2^L}\Psi_L$$

Proof:

By definition of the Walsh transform:

$$\vec{w} = \frac{1}{2^L}\Psi\vec{f}$$

and our earlier observation:

$$\vec{w} = \Psi^{-1}\vec{f}$$

we get:

$$\frac{1}{2^L}\Psi_L = \Psi_L^{-1}$$

□

Observations:

- This means that elements of the matrix Ψ^{-1} are all integer multiples of the inverse of a power of two. Therefore, if the values of a function are all integers, then the Walsh coefficients are all representable exactly in a computer. Also note that, if all the Walsh coefficients are integers then so are the function values.

- From the above theorem, it is clear that

$$\Psi_L^2 = 2^L \mathbf{I}$$

  But this is just a reformulation of the alternative form of the Orthogonality Theorem. In fact the Orthogonality Theorem can be used to prove the last theorem!

- We have shown that the following equations are duals of each other:

$$\Psi \vec{w} = \vec{f}$$

$$\frac{1}{2^L} \Psi \vec{f} = \vec{w}$$

2.7 Walsh Transforms

The linear transformation from a function to its Walsh coefficients can be conveniently treated as a transform function. Let $\mathcal{W}$ be the **Walsh transform**, a function that maps an L bit function to another L bit function via the linear transform induced by Ψ.

$$\mathcal{W}(f) = \frac{1}{2^L} \Psi_L f$$

In this form, $\mathcal{W}(f)$ represents the function that returns the Walsh coefficients of f. Thought of another way, $\mathcal{W}(f)$ in vector form is $\vec{w}^f$. From this definition we know that

$$\mathcal{W}(\mathcal{W}(f)) = \frac{1}{2^L} f$$

Then we can interpret the Product Coefficient Theorem as

$$\mathcal{W}(fg) = \mathcal{W}(f) \circ \mathcal{W}(g)$$

Using this notation. the Walsh transform of a convolution of two functions can be easily computed. This is an analogue of the well known Convolution Theorem from Fourier analysis (Weaver. 1989).

Theorem 22 (Transform of Convolution)

$$2^L \mathcal{W}(f)\mathcal{W}(g) = \mathcal{W}(f \circ g)$$

Proof:

We know that

$$\begin{aligned}
\mathcal{W}(fg) &= \mathcal{W}(f) \circ \mathcal{W}(g) \\
\mathcal{W}(\mathcal{W}(fg)) &= \mathcal{W}(\mathcal{W}(f) \circ \mathcal{W}(g)) \\
\tfrac{1}{2^L} fg &= \mathcal{W}(\mathcal{W}(f) \circ \mathcal{W}(g)) \\
&\quad \text{Let } f = \mathcal{W}(f') \text{ and } g = \mathcal{W}(g') \\
\tfrac{1}{2^L}\mathcal{W}(f')\mathcal{W}(g') &= \mathcal{W}(\mathcal{W}(\mathcal{W}(f')) \circ \mathcal{W}(\mathcal{W}(g')) \\
\tfrac{1}{2^L}\mathcal{W}(f')\mathcal{W}(g') &= \mathcal{W}(\tfrac{1}{2^L} f') \circ \tfrac{1}{2^L} g' \\
\tfrac{1}{2^L}\mathcal{W}(f')\mathcal{W}(g') &= (\tfrac{1}{2^L})^2 \mathcal{W}(f' \circ g') \\
2^L \mathcal{W}(f')\mathcal{W}(g') &= \mathcal{W}(f' \circ g')
\end{aligned}$$

This theorem and the Product Coefficient Theorem gives us four key relations between the product and the convolution operators. The result is the following elegant symmetry:

$$\mathcal{W}(fg) = \mathcal{W}(f) \circ \mathcal{W}(g) \qquad fg = 2^L \mathcal{W}(\mathcal{W}(f) \circ \mathcal{W}(g))$$

$$\mathcal{W}(f \circ g) = 2^L \mathcal{W}(f)\mathcal{W}(g) \qquad f \circ g = 2^{2L} \mathcal{W}(\mathcal{W}(f)\mathcal{W}(g))$$

The development of the previous theorem leads to the following relationship between the dot products of two functions. represented in the function space and the dot product of the functions in Walsh space. The **dot product** of two functions is defined to be the dot product of the functions represented as vectors. This theorem is also known as Parseval's Theorem in Fourier analysis is stated but not proven in (Manela and Campbell. 1992).

Theorem 23 (Dot Product)

Let f and g be two functions then:

$$f \cdot g = 2^L (\mathcal{W}(f) \cdot \mathcal{W}(g))$$

Proof:

$$f \cdot g = \sum_{x=0}^{2^L-1} f(x)g(x)$$

$$= \sum_{x=0}^{2^L-1} (\sum_{j=0}^{2^L-1} w_j^f \psi_j(x))(\sum_{k=0}^{2^L-1} w_k^g \psi_k(x))$$

$$= \sum_{x=0}^{2^L-1} \sum_{j=0}^{2^L-1} \sum_{k=0}^{2^L-1} w_j^f w_k^g \psi_k(x)\psi_j(x)$$

$$= \sum_{j=0}^{2^L-1} \sum_{k=0}^{2^L-1} w_j^f w_k^g \sum_{x=0}^{2^L-1} \psi_k(x)\psi_j(x)$$

By the Orthogonality Theorem. the inner most sum is zero for all values of j and k where $j \neq k$. When $j = k$. then the sum is 2^L. Therefore setting $k = j$:

$$= \sum_{j=0}^{2^L-1} w_j^f w_j^g 2^L$$

$$= 2^L \sum_{j=0}^{2^L-1} w_j^f w_j^g$$

$$= 2^L (\mathcal{W}(f) \cdot \mathcal{W}(g))$$

The Dot Product Theorem reads as follows in terms of functions and Walsh coefficients:

$$\sum_{j=0}^{2^L-1} f(j)g(j) = 2^L \sum_{j=0}^{2^L-1} (w_j^f)(w_j^g)$$

This leads to an interesting observation.

Corollary 23a (Sum of Squares)

$$\sum_{j=0}^{2^L-1} f(j)^2 = 2^L \sum_{j=0}^{2^L-1} (w_j^f)^2$$

This means that if you consider the Walsh transform as a transformation of coordinates, the length of the function vector is always $\sqrt{2^L}$ longer than the Walsh coefficient vector.

2.7.1 Mean and Variance from Transforms

Dot products can be used to sum over all function values. For example, the mean of an L bit function f, denoted $E(f)$ is

$$E(f) = (f \cdot \vec{1})/2^L = \mathcal{W}(f) \cdot \mathcal{W}(\vec{1}) = w_0^f$$

$\mathcal{W}(\vec{1})$ is of course 0 everywhere but in the position corresponding to w_0. Hence, the dot product on the right is simply w_0 of $\mathcal{W}(f)$. This confirms what we proved early in the chapter in the Mean Value Theorem.

The variance of f, denoted $V(f)$ can be computed as

$$\begin{aligned} V(f) &= E(f^2) - (E(f))^2 \\ &= \tfrac{1}{2^L}(f \cdot f) - (w_0^f)^2 \\ &= \tfrac{1}{2^L} 2^L \textstyle\sum_{j=0}^{2^L-1} (w_j^f)^2 - (w_0^f)^2 \\ &= \textstyle\sum_{j=1}^{2^L-1} (w_j^f)^2 \end{aligned}$$

This means if the Walsh transform is treated as a projection, then the standard deviation is the portion of the length of the vector not accounted for by the projection onto v_0, which is just the mean.

2.8 The Fast Walsh Transform

It turns out that the Walsh transform can be quickly computed using an analog of the fast fourier transform. The mechanics of the transform is presented without proof in (Goldberg, 1989c). Here I will use the block matrix form of the Ψ matrix to derive the Fast Walsh Transform. A **Fast Walsh Transform** allows one to compute the product $\Psi_L \vec{v}$ in $O(L2^L)$ operations. This may seem like a lot of computation, but it is a vast improvement over the direct matrix multiply approach which takes $O(2^{2L})$ operations.

Let $\vec{p}$ be defined as

$$\vec{p} = \Psi_L \vec{v} = \begin{bmatrix} A & A \\ A & -A \end{bmatrix} \vec{v}$$

Both vectors $\vec{v}$ and $\vec{p}$ may be blocked in blocks of 2^{L-1} elements giving

$$\begin{bmatrix} A & A \\ A & -A \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} Av_1 + Av_2 \\ Av_1 - Av_2 \end{bmatrix} = \begin{bmatrix} A(v_1 + v_2) \\ A(v_1 - v_2) \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}$$

therefore

$$p_1 = A(v_1 + v_2) \quad \text{and} \quad p_2 = A(v_1 - v_2)$$

where A is Ψ_{L-1}. But Ψ_{L-1} and p_1 can be expressed in block form and the process repeated. This defines a recursive relationship that goes L levels deep, each time subdividing the vector $\vec{v}$. The exact process is best explained with the following C function, which implements $\Psi\vec{v}$ and can be used to efficiently compute **both** transforms: $\vec{w} \rightarrow \vec{f}$ and $\vec{f} \rightarrow \vec{w}$.

```
void fwt(REAL *v, int size)
{
    int s2;
    REAL *a, *b, tmp;

    // recursively handle each half of the vector v
    s2 = size>>1;
    if (s2>1) {
        fwt(v, s2);      // recursively compute A v_1
        fwt(v+s2, s2);   // recursively compute A v_2
    }

    // now create the sum and difference parts for
    // for both halves
    for (a=v, b=v+s2; a<v+s2; a++, b++) {
        tmp = *a;
        *a += *b;          //  p_1 = A (v_1 + v_2)
        *b = tmp - *b;     //  p_2 = A (v_1 - v_2)
    }
}
```

The same technique I used to develop the Fast Walsh Transform can be used to efficiently implement other transforms as we will see later.

2.9 The Vose-Leipens Transform

I cannot leave this chapter without mentioning an alternate linear decomposition to Walsh functions and coefficients that I call the **Vose-Leipens (VL) Transform** after the inventors. Although this transform seems simple and elegant, we will see that it lacks several important features that the Walsh transform offers.

In the Vose-Leipens transform the value of a function f can be computed as a weighted

sum of functions:

$$f(x) = \sum_{i=0}^{2^L-1} w^{vl} \psi_i^{vl}(x)$$

where $\psi_i^{vl}(x) = \begin{cases} 0 & \text{if } i \not\subseteq x \\ 1 & \text{if } i \subseteq x \end{cases}$

In this case each w_i^{vl} represents the contribution added by the nonlinearity introduced by the 1 bits in i. For example if $L = 3$ then

$$f(5) = w_5^{vl} + w_4^{vl} + w_1^{vl} + w_0^{vl}$$

while from the Walsh transform we get

$$f(5) = w_7 - w_6 + w_5 - w_4 - w_3 + w_2 - w_1 + w_0$$

Using the Vose-Leipens transform it is easy and intuitive to isolate the effects of bit interactions. It is also easy to compute the effects of N bit interactions. In the above example w_5^{vl} is the effect of interactions of bits 0 and 2. On the other hand, it is difficult and inefficient to compute hyperplane averages using Vose-Leipens transform. The Vose-Leipens transform also lacks the positive and negative cancellation which yielded the Orthogonality Theorem. This was critical in many of the proofs.

In the Vose-Leipens transform, there is a parallel for the Ψ matrix I will call Ψ^{vl} matrix. Again, lines are included to show symmetries.

$$\Psi_3^{vl} = \left[\begin{array}{cc|cc|cccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline\hline 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right]$$

In blocked matrix format, this has the recursive form:

$$\vec{p} = \Psi_L^{vl} = \begin{bmatrix} A & 0 \\ A & A \end{bmatrix}$$

where $A = \Psi_{L-1}^{vl}$. We lose three very important properties we had with the Ψ matrices of Walsh transformation, namely: the matrix is **not** symmetric, it does not form an orthogonal basis, and the inverse is not a constant times the matrix itself. The inverse is

$$\vec{p} = \Psi_L^{vl\ -1} = \begin{bmatrix} A^{-1} & 0 \\ -A^{-1} & A^{-1} \end{bmatrix}$$

From Ψ^{vl} and its inverse it is easy to derive conversions between function space, VL and Walsh transforms. A corresponding **Fast VL Transform** algorithm can be easily derived using the block matrix technique we used to develop the Fast Walsh Transform.

2.10 Summary

In this chapter we covered the basics of Walsh analysis in detail. Walsh functions were presented and techniques developed for using them. We saw that functions could be decomposed into a weighted sum of Walsh functions and that these Walsh functions formed an orthogonal basis. From this came Walsh analysis which could be viewed from the perspective of polynomials, matrices, or transforms. Walsh coefficients were easy to compute conceptually but exponential time is required even with the Fast Walsh Transform. What will be needed to make computing Walsh coefficients practical is an understanding of what functions have a very limited or highly patterned set of Walsh coefficients. In later chapters, I will present several important and common classes of problems with low epistasis.

In the next chapter we look at the concept of hyperplanes in the domain. This will help us understand the underlying structure of functions in a way that is directly related to the way evolutionary algorithms probe the domain space.

Chapter 3

Hyperplanes and Function Embedding

Evolutionary algorithms use mating and mutating to generate new points in the search space to be sampled. These work by cutting and splicing subsets of bits. Furthermore, many of the major tenants of genetic algorithms such as the Building Block Hypothesis (Goldberg, 1989c) and the Schema Theorem (Holland, 1975) are based on these mechanistically derived subsets of bits. For example, the Schema theorem suggests that it is the subsets of bits that compete for dominance in the population. The Building Block Hypothesis suggests that it is subsets of bits that are assembled from parents to make better children. Recent research on hyperplane ranking (Whitley et al., 1995a; Heckendorn et al., 1997) support the idea that subsets of bits may themselves compete for survival. Developing the mathematics that could measure epistasis for subsets of bits seems relevant to the mechanical theory behind evolutionary algorithms. This was the theme of Bethke's thesis (Bethke, 1981) and extended by others (Goldberg, 1989c; Davidor, 1991; Reeves and Wright, 1995b).

In the last chapter we built up the basic mathematical framework for Walsh analysis. In this chapter, I extend Walsh analysis to study subsets of bits by developing the idea of hyperplanes, which are lower dimensional subcubes of the bit space domain. They can also be thought of as subsets of possible chromosomes with fixed values for a specific subset of their bit positions. I use Walsh coefficients to compute the average value of a function over a hyperplane, a well known result, and develop a Fast Hyperplane Averaging algorithm which I believe to be new. I discuss the partitioning of the domain of a problem by hyperplanes

and introduce a scheme for numbering them that will prove a useful shorthand. Also, I will present a very important pair of functions *pack* and *unpack* that can be used to change the dimensionality of a function. These functions are then used to embed lower dimensional functions in higher dimensional space. The idea of function embedding will become a major theme in the remainder of the dissertation.

3.1 Preliminaries

Consider a bit space $\mathcal{B}^L$ which is the domain of the function we wish to optimize. It is essentially an L dimensional hypercube with the vertices of the cube being the strings in the domain. A subcube of strings of dimension $M : M < L$ can be selected by fixing the values of $L - M$ of the coordinates. This subset of strings forms a **hyperplane** which is an M dimensional slice from the cube containing 2^M strings. The entire domain can be partitioned into 2^{L-M} non-overlapping hyperplanes whose union is the entire hypercube.

There is a classic notation for representing hyperplanes and partitions. A **hyperplane** can be represented as one of the 3^L strings of 0's, 1's and *'s. The 0's and 1's indicate **fixed bit positions**. The all the strings in the hyperplane must have the value indicated by bits in the fixed bit positions. The *'s represent either a 0 or a 1 in the **variable bit positions**. This is like the "don't care" in digital design. There is no required value for the variable bit positions in the strings in the hyperplane. For example the hyperplane $H = 0*1101*$ is a set of four strings:

Hyperplane specification:	0*1101*
	0011010
Strings in the hyperplane:	0011011
	0111010
	0111011

The bits indicated in bold are the ones in the variable bit positions.

The **order of a hyperplane** h with C fixed bit positions is C and is denoted by $o(h)$. A hyperplane of order C defines a set of $2^{(L-C)}$ strings where all possible replacements of the *'s have been defined. The number of strings in h is denoted $|h|$. For example, let's again take hyperplane $H = 0*1101*$ above. We find it is an order 5 hyperplane ($o(H) = 5$)

containing 4 strings ($|H| = 4$).

A **partition** is a set of hyperplanes that all share the same set of fixed bit positions. Therefore, a partition can be specified by a string with a `b` in positions of fixed bits and `*`'s in all other positions. For example, `*b**` represents a partition that contains the two hyperplanes `*1**` and `*0**`. The hyperplane H above is one of 32 hyperplanes in the partition `b*bbbb*`. In general, a partition with M `b`'s defines a set of 2^M nonintersecting hyperplanes, each composed of 2^{L-M} strings whose union is all of the strings in the domain. The **order of a partition** with exactly C `b`'s is C and defines a set of 2^C hyperplanes each of order C. For partition π, the order of π is denoted by $o(\pi)$ and the number of hyperplanes in π is denoted $|\pi|$. The **partition mask** is the bitstring composed by replacing each `b` with a `1` and each `*` with a `0`. For example, the partition mask for `b*bbbb*` is `1011110`.

3.2 Hyperplanes and Walsh Functions

In order to prove theorems about hyperplanes we need two functions α and β that are defined on hyperplanes as per Goldberg (Goldberg, 1989a). If $\mathcal{H} = \{0, 1, *\}$ and $h \in \mathcal{H}^L$, α and β can be defined on a hyperplane as $\alpha, \beta : \mathcal{H}^L \rightarrow \mathcal{B}^L$:

$$\alpha(h)[i] = \begin{cases} 0 & \text{if } h[i] = * \\ 1 & \text{if } h[i] = 0 \text{ or } 1 \end{cases} \qquad \beta(h)[i] = \begin{cases} 0 & \text{if } h[i] = * \text{ or } 0 \\ 1 & \text{if } h[i] = 1 \end{cases}$$

In summary: α is a mask that is 1 where there are constant bit positions. β is a mask that is 1 where there are 1's in the fixed bit positions and 0 elsewhere. For example, for the hyperplane H = `0*1101*` : $\alpha(H)$ = `1011110` and $\beta(H)$ = `0011010`. Notice that $\alpha(H)$ gives the partition mask for the partitioning that contains H.

Observations:

- For hyperplane h and a string $x : x \in h$:

$$\beta(h) \subseteq \alpha(h)$$

$$(x \wedge \alpha(h)) = \beta(h)$$

$$bc(\alpha(h)) = o(h)$$

$$bc(x) \geq bc(\beta(h))$$

- If h_x is the hyperplane containing **only** the single string x and H is the hyperplane containing the whole domain then

$$\beta(h_x) = x$$

$$\alpha(h_x) = \vec{1}$$

$$\beta(H) = \vec{0}$$

$$\alpha(H) = \vec{0}$$

- Specifying both $\alpha(h)$ and $\beta(h)$ is sufficient to uniquely specify the hyperplane. As we will see later, this will be the basis of hyperplane numbering.

A generalization of the very useful Balanced Sum Theorem exists for hyperplanes. It shows that under certain conditions the sum over all strings in a hyperplane of a given Walsh function is zero. This theorem is useful for eliminating sums. This theorem was indirectly discussed and a logical argument given for it in the proof of a different theorem in (Goldberg, 1989a).

Theorem 24 (Balanced Sum for Hyperplanes)

$$\sum_{x \in h} \psi_j(x) = \begin{cases} 0 & \text{if } j \not\subseteq \alpha(h) \\ \psi_j(\beta(h))|h| & \text{if } j \subseteq \alpha(h) \end{cases}$$

Proof:

CASE 1: if $j \subseteq \alpha(h)$

$$\sum_{x \in h} \psi_j(x) \quad = \sum_{x \in h} \psi_{j \wedge \alpha(h)}(x)$$

by using the Associative Masking Theorem:

$$= \sum_{x \in h} \psi_j(x \wedge \alpha(h))$$

$$= \sum_{x \in h} \psi_j(\beta(h))$$

$$= |h| \psi_j(\beta(h))$$

CASE 2: if $j \not\subseteq \alpha(h)$ (we will proceed as for the Balanced Sum Theorem) then $\exists k$ such that $j[k] = 1$ and k is a bit position outside the fixed bit positions of h. We now divide the $2^{L-|h|}$ x values into 2 sets.

Let A be the set of all $\psi_j(x)$ where $x[k] = 1$.

Let B be the set of all $\psi_j(x)$ where $x[k] = 0$.

If $x \in h$, then $x \ominus 2^k \in h$, since the k^{th} bit position is in the nonconstant portion of the bitstrings in h. Furthermore, if $x \in A$, then $x \ominus 2^k \in B$. Let $y = x \ominus 2^k$. Note that for every $x \in A$ there is a unique $y \in B$ and vice versa. Therefore the $|A| = |B|$.

The remainder of the proof proceeds as in the Balanced Sum Theorem except that A and B may be smaller sets contained entirely in a hyperplane.

$$\psi_j(y) \quad = Y(\bigoplus_{i=0}^{L-1}(y[i] \wedge j[i]))$$

For the k^{th} term in the above exclusive-or:

$$(y[k] \wedge j[k]) \quad = ((x[k] \ominus 1) \wedge j[k])$$

$$(x[k] \wedge j[k]) \ominus 1)$$

Bringing the 1 out of the exclusive-or:

$$\begin{aligned}\psi_j(x) &= Y^{\ominus}(1 \ominus (\bigoplus_{i=0}^{L-1}(x[i] \wedge j[i]))) \\ &= Y^{\ominus}(1) Y^{\ominus}(\bigoplus_{i=0}^{L-1}(x[i] \wedge j[i]))) \\ &= -1\psi_j(y)\end{aligned}$$

Since there is a one to one onto correspondence between elements of A and elements of B and each pair is of opposite sign the sum is zero.

3.3 Hyperplane Averaging

The **hyperplane average** is the average value of all of the function values for strings in the hyperplane. The average of function f over hyperplane h is denoted:

$$\langle f \rangle_h = \frac{1}{|h|} \sum_{x \in h} f(x)$$

The hyperplane average is the expected function value for a string randomly selected from the hyperplane with uniform probability. When hyperplanes compete for their place in the population, it is this sampled value that is believed to represent the fitness of the schema. Therefore, a hyperplane average can be thought of as the **fitness of the hyperplane**. The following theorem is a well known result and this proof follows closely the lines of Goldberg's proof (Goldberg, 1989a).

Theorem 25 (Hyperplane Averaging)

$$\langle f \rangle_h = \sum_{j : j \subseteq \alpha(h)} w_j \psi_j(\beta(h))$$

Proof:

We begin by substituting the inverse Walsh transform of the function to be averaged:

$$\begin{aligned}
\frac{1}{|h|}\sum_{x\in h} f(x) &= \frac{1}{|h|}\sum_{x\in h}\sum_{j=0}^{2^L-1} w_j\psi_j(x) \\
&= \frac{1}{|h|}\sum_{j=0}^{2^L-1} w_j \sum_{x\in h}\psi_j(x) \\
&\quad \text{Only when } j \subseteq \alpha(h) \text{ is the inner sum nonzero} \\
&= \frac{1}{|h|}\sum_{j:j\subseteq\alpha(h)} w_j|h|\psi_j(\beta(h)) \\
&= \sum_{j:j\subseteq\alpha(h)} w_j\psi_j(\beta(h))
\end{aligned}$$

□

The Hyperplane Averaging Theorem says that we can compute the average value of a function across all strings in a hyperplane by looking at only the Walsh coefficients w_j where j has its 1 bits contained in the fixed bits of the hyperplane. This means that if only a few bit positions are fixed bits, then the average value of the function over a hyperplane can be computed more quickly by summing Walsh coefficients than by summing the function values. This, of course, assumes that the Walsh coefficients are available. However, if the hyperplane has more than half of the bit positions fixed, then this approach becomes less efficient than enumeration.

Corollary 25a (Sufficiency for Hyperplane Averages)

The hyperplane fitness of hyperplane h can be computed using only those Walsh coefficients w_j where $bc(j) \leq o(h)$

Proof:

From the Hyperplane Averaging Theorem, we know that:

$$\frac{1}{|h|}\sum_{x\in h} f(x) = \sum_{j\subseteq\alpha(h)} w_j\psi_j(\beta(h))$$

For all $j \subseteq \alpha(h)$, we see that $bc(j) \leq bc(\alpha(h))$. But we also know that $bc(\alpha(h)) = o(h)$. Therefore, only w_j where $bc(j) \leq o(h)$ are used to compute the hyperplane average.

□

3.4 Examples of Hyperplane Averages

Table 3.1 illustrates several points about the Hyperplane Averaging Theorem. First, the selection of Walsh coefficients used to compute the average fitness for a hyperplane depends on $\alpha(h)$. This is because the indices of the coefficients are exactly those strings j such that $j \subseteq \alpha(h)$. The signs of the coefficients are based on $\beta(h)$.

Table 3.1: Examples of Computing the Average Fitness of a Hyperplane Using Walsh Coefficients

h	$o(h)$	$\alpha(h)$	$\beta(h)$	average fitness of h
***	0	000	000	w_0
**1	1	001	001	$w_0 - w_1$
1	1	010	010	$w_0 - w_2$
1**	1	100	100	$w_0 - w_4$
0**	1	100	000	$w_0 + w_4$
*00	2	011	000	$w_0 + w_1 + w_2 + w_3$
*01	2	011	001	$w_0 - w_1 + w_2 - w_3$
*10	2	011	010	$w_0 + w_1 - w_2 - w_3$
*11	2	011	011	$w_0 - w_1 - w_2 + w_3$
0*0	2	101	000	$w_0 + w_1 + w_4 + w_5$
101	3	111	101	$w_0 - w_1 + w_2 - w_3 - w_4 + w_5 - w_6 + w_7$

We also can see that $2^{o(h)}$ Walsh coefficients are needed to compute the hyperplane average using the Hyperplane Fitness Theorem since only the $2^{o(h)}$ strings satisfy the $j \subseteq$

$\alpha(h)$ requirement. Since $2^{(L-o(h))}$ function values are needed to compute the hyperplane average directly from the function, there is a tradeoff between computing the hyperplane average by using Walsh coefficients and using function values. Obviously, if $o(h) > L/2$ using the function table would be better, while if $o(h) < L/2$ using the Walsh coefficients would be better. This is easy to see in Table 3.1. When there is only one fixed bit, the hyperplanes consist of 4 strings each, but only 2 Walsh coefficients are necessary to compute their hyperplane average. However, when there are 2 fixed bits then there are only 2 strings in each hyperplane but, it takes 4 Walsh coefficients to compute the hyperplane average for those hyperplanes. So it is clear that for some hyperplanes the Walsh coefficients can be used to compute the hyperplane average more quickly than by summing the function values themselves provided the Walsh coefficients are already available.

For certain broad classes of functions, we will see that having the Walsh coefficients available is polynomially easy and that many of the Walsh coefficients are zero. This will greatly extend the usefulness of this result.

3.5 Walsh Averaging

For pedagogical reasons I will take this opportunity to stress the duality between the table of Walsh coefficients and the table of function values.

Theorem 26 (Walsh Averaging)

$$\frac{1}{|h|}\sum_{x\in h} w_x = \frac{1}{2^L}\sum_{j\subseteq o(h)} f(j)\psi_j(\beta(h))$$

Proof:

$$\begin{aligned}
\frac{1}{|h|}\sum_{x\in h} w_x &= \frac{1}{|h|}\sum_{x\in h}\frac{1}{2^L}\sum_{j=0}^{2^L-1} f(j)\psi_x(j) \\
&= \frac{1}{|h|}\frac{1}{2^L}\sum_{j=0}^{2^L-1} f(j)\sum_{x\in h}\psi_x(j) \\
&= \frac{1}{|h|}\frac{1}{2^L}\sum_{j=0}^{2^L-1} f(j)\sum_{x\in h}\psi_j(x) \\
&= \frac{1}{|h|}\frac{1}{2^L}\sum_{j\subseteq\alpha(h)} f(j)|h|\psi_j(\beta(h)) \\
&= \frac{1}{2^L}\sum_{j\subseteq\alpha(h)} f(j)\psi_j(\beta(h))
\end{aligned}$$

□

3.6 Spectral Decomposition

It will prove a useful concept to do a more coarse breakdown of a function into epistatic subfunctions where the i^{th} subfunction represents all of the i bit epistasis. Therefore, any L bit function, f, can be broken down into a sum of $L+1$ functions S_i such that:

$$f(x) = \sum_{i=0}^{2^L-1} w_i\psi_i(x) = \sum_{i=0}^{L} S_i(x)$$

where

$$S_i(x) = \sum_{j:bc(j)=i} w_j\psi_j(x)$$

The notation in the sum above means the sum over all indices with exactly i bits set to 1. The function S_i is the i^{th} **spectral function**, and the decomposition is called the **spectral decomposition** of f.

We will use the spectral decomposition not only in the following theorem but to discuss function parity and some epistatic measures later in the dissertation.

Theorem 27 (Hyperplane Averages of Spectral Functions)

If $f(x) = S_k(x)$. then the hyperplane average $\langle f \rangle_h = 0$ for all hyperplanes h such that $o(h) < k$.

Proof:

$$\begin{aligned}
\tfrac{1}{|h|}\textstyle\sum_{x\in h} f(x) &= \tfrac{1}{|h|}\textstyle\sum_{x\in h}\sum_{j=0}^{2^L-1} w_j\psi_j(x) \\
&\qquad \text{then since } \; f = S_k \\
&= \tfrac{1}{|h|}\textstyle\sum_{x\in h}\sum_{j:bc(j)=k}^{2^L-1} w_j\psi_j(x) \\
&= \tfrac{1}{|h|}\textstyle\sum_{j:bc(j)=k}^{2^L-1} w_j \sum_{x\in h}\psi_j(x)
\end{aligned}$$

but $\sum_{x\in h}\psi_j(x) \neq 0$ only if $j \subseteq \alpha(h)$. However. since $bc(\alpha(h)) = o(h)$ and $o(h) < k$. $\alpha(h)$ must have fewer than k one bits but j must have exactly k one bits. Therefore $j \not\subseteq \alpha(h)$.

3.7 The Packing Functions

Another important concept is the packing and unpacking of bitstrings by a mask. This will be used later to change the dimensionality of functions and to define a numbering scheme for hyperplanes.

A function *pack* is defined as $pack : \mathcal{B}^L \times \mathcal{B}^L \rightarrow \mathcal{B}^M$ where $M \leq L$. $pack(x.m)$ takes the bits in x and masks them with an L bit mask m such that $bc(m) = M$. The bits selected by the mask are then right justified with zero fill in the result. For example:

$$pack(10101.01101) \Longrightarrow 011$$

A function *unpack* is defined as $unpack : \mathcal{B}^M \times \mathcal{B}^L \rightarrow \mathcal{B}^L$ where $M \leq L$. $unpack(x.m)$ takes all the bits in x and unpacks them. one bit in each position that corresponds to a 1 in m. where $m : bc(m) = M$. All unset bits remain 0. For example:

$$unpack(011.01101) \Longrightarrow 00101$$

What follows is a list of observations about *pack* and *unpack*. The first two observations point out that *pack* and *unpack* are not strict inverses of one another. Note that *pack* destroys information about the original argument.

Observations:

- $unpack(pack(j.m).m) = j \wedge m$
- $pack(unpack(j.m).m) = j$
- $unpack(x.m) \subseteq m \qquad \forall x \in \mathcal{B}^{bc(m)}$
- $bc(pack(j.m)) = bc(j \wedge m)$
- $bc(unpack(j.m)) = bc(j)$
- $pack(i \wedge j.m) = pack(i.m) \wedge pack(j.m)$
- Given $m \in \mathcal{B}^L$, $j \in \mathcal{B}^M$, the set of $x \in \mathcal{B}^L$ such that $pack(x.m) = j$ is exactly the set of strings in $\mathcal{B}^L$ in an order M hyperplane h where $\alpha(h) = m$. The choice of j and m selects the particular order M hyperplane.

Although it may not be apparent at first, the Pack/Unpack Equivalency Theorem, which follows, will give us the very important ability to change the dimension of the domain of a function. This is because the Walsh function on the left of the equation can be operating on different length bit strings than the function on the right. Specifically, the Walsh function on the left hand side in the statement of the theorem is in $\mathcal{B}^L$, while the Walsh function on the right side is in $\mathcal{B}^M$.

Theorem 28 (Pack/Unpack Equivalency)

For $i \in \mathcal{B}^M$ and $j \in \mathcal{B}^L$ with $M \leq L$ and a mask $m \in \mathcal{B}^L$ with $bc(m) = M$:

$$\psi_j(unpack(i.m)) = \psi_{pack(j.m)}(i)$$

Proof:

$$\begin{aligned}
\psi_j(unpack(i.m)) &= Y(\bigoplus_{n=0}^{L-1} unpack(i.m)[n] \wedge j[n]) \\
&= Y(\bigoplus_{n=0}^{L-1}(unpack(i.m) \wedge j)[n]) \\
&\quad \textit{pack} \text{ with mask } m \text{ right justifies bits unpacked} \\
&\quad \text{with mask } m. \text{ so the parity in xor is the same:} \\
&= Y(\bigoplus_{n=0}^{M-1} pack(unpack(i.m) \wedge j).m)[n]) \\
&= Y(\bigoplus_{n=0}^{M-1}(pack(unpack(i.m).m) \wedge pack(j.m))[n]) \\
&= Y(\bigoplus_{n=0}^{M-1}(i \wedge pack(j.m))[n]) \\
&= Y(\bigoplus_{n=0}^{M-1} i[n] \wedge pack(j.m)[n]) \\
&= \psi_{pack(j.m)}(i)
\end{aligned}$$

Another useful theorem for changing the dimension of an expression involving Walsh functions is the following corollary. Notice that on the left hand side of the statement of the corollary $k.m \in \mathcal{B}^L$ while on the right the results of the *pack* function calls are in $\mathcal{B}^M$.

Corollary 28a

$$\psi_j(k \wedge m)) = \psi_{pack(j.m)}(pack(k.m))$$

Proof:

We begin by substituting $pack(k.m)$ for i in the Pack/Unpack Equivalency theorem.

$$\begin{aligned}
\psi_j(unpack(i.m)) &= \psi_{pack(j.m)}(i) \\
\psi_j(unpack(pack(k.m).m)) &= \psi_{pack(j.m)}(pack(k.m)) \\
\psi_j(k \wedge m)) &= \psi_{pack(j.m)}(pack(k.m))
\end{aligned}$$

3.8 Hyperplane Numbering

It was noted earlier that $\alpha(h)$ can be used to decide how the domain is partitioned. $\beta(h)$ can then be used to select a hyperplane from the partition. This was especially evident in the formula for hyperplane averages. It is clear, therefore, that a unique numbering for hyperplanes can be established based on α and β. To make this more formal, define the **partition mask** as a binary string by replacing the b's by 1's and *'s by 0's. Let the partition mask be the string m. We use $h_{m,n}$. $m \in \mathcal{B}^L$. $bc(m) = M$. $M \leq L$. $n \in \mathcal{B}^M$ to denote an M order hyperplane in $\mathcal{B}^L$ such that $x \in h_{m,n}$ if and only if $pack(x, m) = n$. In this case, the partition mask m selects how the domain is partitioned and the **hyperplane number**, n, selects one of the hyperplanes from the partition.

For example: if $h_{m,n}$ specifies a hyperplane in $\mathcal{B}^7$ and if n = 10110 and m = 1101110 then $h_{m,n}$ is a 5^{th} order hyperplane contained in the partition bb*bbb*. m specified that partition bb*bbb* and n specifies the values of the fixed bit positions, 10110. Therefore $h_{m,n}$ is 10*110*.

The partitioning of the domain space by partition mask m is denoted $h_{m,*}$. The partitioning can be viewed as a concatenation of $2^{bc(m)}$ hyperplanes $h_{m,n}$ represented as row vector of column vectors:

$$\begin{bmatrix} h_{m,0} & h_{m,1} & h_{m,2} & \cdots & h_{m,2^{bc(m)}-1} \end{bmatrix} \tag{3.1}$$

This forms a $2^{bc(\overline{m})} \times 2^{bc(m)}$ matrix H such that if $H_{i,n} = x$ then $i = pack(x, \overline{m})$ and $n = pack(x, m)$, where $\overline{m}$ is the complement of m. For example, a function in $\mathcal{B}^5$ with a partition of *b*bb would give m = 01011. $\overline{m}$ = 10100 and a domain matrix of:

$$
\begin{array}{c}
 \\
\uparrow \\
pack(x.10100) \\
\downarrow
\end{array}
\begin{array}{c}
\longleftarrow pack(x.01011) \longrightarrow \\
\left[\begin{array}{cccccccc}
00000 & 00001 & 00010 & 00011 & 01000 & 01001 & 01010 & 01011 \\
00100 & 00101 & 00110 & 00111 & 01100 & 01101 & 01110 & 01111 \\
10000 & 10001 & 10010 & 10011 & 11000 & 11001 & 11010 & 11011 \\
10100 & 10101 & 10110 & 10111 & 11100 & 11101 & 11110 & 11111
\end{array}\right]
\end{array}
\tag{3.2}
$$

In this matrix the bold bits $\{\mathbf{0.1}\}$ are those determined by $pack(x.\overline{m})$. All other bits have values determined by $pack(x.m)$. It is important to notice that $\alpha(h_{m.n}) = m$ is a constant for a given partitioning of the domain while $\beta(h_{m.n}) = unpack(n.m)$ is unique for each hyperplane in the domain. This leads to these observations about numbered hyperplanes:

Observations:

Given that a string $x \in h_{m.n}$:

- $\alpha(h_{m.n}) = m$
- $\beta(h_{m.n}) = unpack(n.m)$
- $|h_{m.n}| = 2^{L-bc(m)}$
- $pack(x.m) = n$
- $x \wedge m = unpack(n.m) = \beta(h_{m.n})$

Theorem 29 (Balanced Sum for Numbered Hyperplanes)

For L bit functions:

$$
\sum_{x \in h_{m.n}} \psi_j(x) = \begin{cases} 0 & \text{if } j \not\subseteq m \\ \psi_j(unpack(n.m))2^{L-bc(m)} & \text{if } j \subseteq m \end{cases}
$$

Proof:

This is easily proven given the Balance Sum Theorem for Hyperplanes:

$$\sum_{x \in h} \psi_j(x) = \begin{cases} 0 & \text{if } j \not\subseteq \alpha(h) \\ \psi_j(\beta(h))|h| & \text{if } j \subseteq \alpha(h) \end{cases}$$

and setting $h = h_{m,n}$. We then substitute the observations about numbered hyperplanes above.

We are now in position to prove a hyperplane averaging theorem for numbered hyperplanes. But this comes with a twist. The second part of the theorem contains a sum that is over possibly shorter strings. To emphasize this, I have noted the dimension of the indices of the sums in the two equations in the theorem statement. This is our first theorem that allows us to change the dimension of the domain of the functions we are examining. We will put this to use in creating a Fast Hyperplane Averaging algorithm.

Theorem 30 (Hyperplane Averaging for Numbered Hyperplanes)

$$\langle f \rangle_{h_{m,n}} = \sum_{j \in m} w_j \psi_j(unpack(n, m)) \qquad \text{where } j \in \mathcal{B}^L \tag{3.3}$$

or

$$\langle f \rangle_{h_{m,n}} = \sum_{k=0}^{2^{bc(m)}} w_{unpack(k,m)} \psi_k(n) \qquad \text{where } k \in \mathcal{B}^{bc(m)} \tag{3.4}$$

Proof:

The first equation is proved by a simple substitution into the Hyperplane Averaging Theorem of $\alpha(h_{m,n}) \leftarrow m$ and $\beta(h_{m,n}) \leftarrow unpack(n, m)$.

To prove the second part we begin with the first and apply the Pack/Unpack Equivalency Theorem:

$$\langle f \rangle_{h_{m,n}} = \sum_{j \in m} w_j \psi_{pack(j,m)}(n) \tag{3.5}$$

Now the Walsh function is in the dimension of n. $n \in 2^{bc(m)}$. Let $k = pack(j, m)$. Since $j \subseteq m$. it must be the case that $unpack(k, m) = j$. Now I substitute for both j and $pack(j, m)$ in Equation 3.5. The range of the sum becomes a simple enumeration and the theorem is proved.

$$\langle f \rangle_{h_{m,n}} = \sum_{k=0}^{2^{bc(m)}} w_{unpack(k,m)} \psi_k(n)$$

□

In the next section. we will apply this theorem.

3.9 Fast Hyperplane Averaging

Consider any hyperplane in the partition created by mask m. that is $h_{m,*}$. Notice the Walsh coefficients in the sum in Equation 3.4 are dependent only on m. The important observation here is that the Walsh coefficients used are the same for computing the hyperplane average of any hyperplane $h_{m,*}$. All that changes are the values returned by the Walsh functions. In fact. it is clear by inspection that the sum is performing a Walsh transform on a reduced set of Walsh coefficients!

We can use this to derive a simple algorithm for computing the hyperplane averages of **all** of the hyperplanes in a given partition simultaneously. This function is particularly useful when we want to examine all the hyperplane averages in a given partition to see how they might be ranked or compete by fitness. First. form a new function by extracting all of the Walsh coefficients whose indices are contained in the partition mask. Then perform a Walsh transform on the result. The vector you get is the hyperplane averages for all of the hyperplanes in the partition. The C++ code below implements this function.

```
REAL *hyperplaneAvgs(REAL *w, int size, UINT mask)
```

```
{
    int i;
    REAL *subW;
    int subSize;

    subSize = 1<<bitCount(mask);      // determine sub array size
    subW = new REAL [subSize];        // allocate space for array

    // collect the relevant Walsh coefficients
    for (i=0; i<subSize; i++) {
        subW[i] = w[unpack(i, mask)];
    }

    // now transform them using the fast Walsh transform
    fwt(subW, subSize);

    // return the array of hyperplane averages
    return subW;
}
```

In this function `w` points to an array of all the Walsh coefficients. `Size` contains the number of elements in the function. `Mask` is the partition mask that will create the set of hyperplanes to be averaged. If π is a partition, this algorithm computes the $2^{o(\pi)}$ hyperplane averages in just $O(2^{o(\pi)}o(\pi))$ time. Notice that if `mask` $= \vec{1}$, that is, is all 1's, this function simply clones the function vector and performs an inverse Walsh transform yielding the original function values. What it has done is compute the average of all hyperplanes of size 1 element.

3.10 Function Embedding

We can also use *pack* and *unpack* to **embed** a lower dimensional function in a higher one. This is a very powerful concept in that it lets us compose complex functions from sums of smaller functions whose arguments involve only limited portions of the chromosome. This might occur in real world problems whenever you have a set of objects that interact in limited ways and each interaction contributes to the fitness function in a linear way. An example of such an interaction is a shipping problem where there is a value associated not only with each shipping terminal but perhaps also with the interaction between two terminals when products are shipped between them. This problem is suggestive of any kind of problem that can be modeled as a graph where the value of problem to be optimized is not only dependent on the nodes but the arcs as well.

Let's start by defining an embedded function. Let $g : \mathcal{B}^M \rightarrow \mathcal{R}$ and $f : \mathcal{B}^L \rightarrow \mathcal{R}$ with $L \geq M$. f is said to be an **embedding** of g if there is a mask $m : bc(m) = M$ and $m \in \mathcal{B}^L$ such that

$$f(x) = g(pack(x, m)) \qquad \forall x \in \mathcal{B}^L$$

In short, to embed a lower dimensional function, g, in a higher dimension function, f, using mask m, set the fitness of every string in the hyperplane $h_{m,x}$ of f to $g(x)$. m is referred to as the **embedding mask**. We denote f as an embedding of g using mask m by:

$$f = \mathcal{E}(g, m)$$

The assignment of values to the various hyperplanes is best illustrated using the same domain matrix we used when we first explained hyperplane numbering (see eq. 3.2).

$$
\begin{array}{c}
 \\
\uparrow \\
pack(x.\overline{m}) \\
\downarrow
\end{array}
\begin{array}{c}
\longrightarrow\ pack(x.m)\ \longrightarrow \\
\left[\begin{array}{ccccc}
g(0) & g(1) & g(2) & \ldots & g(2^{bc(m)}-1) \\
g(0) & g(1) & g(2) & \ldots & g(2^{bc(m)}-1) \\
\vdots & \vdots & \vdots & & \vdots \\
g(0) & g(1) & g(2) & \ldots & g(2^{bc(m)}-1)
\end{array}\right]
\end{array}
\tag{3.6}
$$

It turns out that the nonzero Walsh coefficients for a function and an embedding of that function are the same.

Theorem 31 (Embedding)

Let $g : \mathcal{B}^M \rightarrow \mathcal{R}$. $f : \mathcal{B}^L \rightarrow \mathcal{R}$. and $f = \mathcal{E}(g.m)$ then

$$
w_i^f = \begin{cases} w^g_{pack(i.m)} & \text{if} \quad i \subseteq m \\ 0 & \text{if} \quad i \not\subseteq m \end{cases}
$$

where w^g and w^f are Walsh coefficients for the g and f functions respectively and $i.m \in \mathcal{B}^L : bc(m) = M$.

Proof:

$$
\begin{aligned}
w_i^f &= \tfrac{1}{2^L}\textstyle\sum_{x=0}^{2^L-1} f(x)\psi_x(i) \\
&= \tfrac{1}{2^L}\textstyle\sum_{x=0}^{2^L-1} g(pack(x,m))\psi_x(i) \\
&\qquad \text{we can now convert the sum into two sums by summing the elements of each of the } 2^M \text{ hyperplanes in the partition } h_{m,*} \text{ separately.} \\
&= \tfrac{1}{2^L}\textstyle\sum_{j=0}^{2^M-1}\sum_{x:pack(x,m)=j} g(pack(x,m))\psi_x(i) \\
&= \tfrac{1}{2^L}\textstyle\sum_{j=0}^{2^M-1}\sum_{x:pack(x,m)=j} g(j)\psi_x(i) \\
&= \tfrac{1}{2^L}\textstyle\sum_{j=0}^{2^M-1} g(j)\sum_{x:pack(x,m)=j}\psi_x(i) \\
&= \tfrac{1}{2^L}\textstyle\sum_{j=0}^{2^M-1} g(j)\sum_{x\in h_{m,j}}\psi_x(i) \\
&= \tfrac{1}{2^L}\textstyle\sum_{j=0}^{2^M-1} g(j)\sum_{x\in h_{m,j}}\psi_i(x)
\end{aligned}
$$

We know from the Balanced Sum for Numbered Hyperplanes Theorem that

$$
\sum_{x\in h_{m,n}}\psi_j(x) = \begin{cases} 0 & \text{if } j \not\subseteq m \\ \psi_j(unpack(n,m))2^{L-bc(m)} & \text{if } j \subseteq m \end{cases}
$$

So, starting where we left off:

CASE 1: Assume $i \not\subseteq m$:

$$
\begin{aligned}
w_i^f &= \tfrac{1}{2^L}\textstyle\sum_{j=0}^{2^M-1} g(j)\sum_{x\in h_{m,j}}\psi_i(x) \\
&= \tfrac{1}{2^L}\textstyle\sum_{j=0}^{2^M-1} g(j)0 \\
&= 0
\end{aligned}
$$

CASE 2: Assume $i \subseteq m$:

$$
\begin{aligned}
w_i^f &= \tfrac{1}{2^L} \textstyle\sum_{j=0}^{2^M-1} g(j) \sum_{x \in h_{m,j}} \psi_i(x) \\
&= \tfrac{1}{2^L} \textstyle\sum_{j=0}^{2^M-1} g(j) \psi_i(unpack(j, m)) 2^{L-bc(m)} \\
&= \tfrac{1}{2^L} \textstyle\sum_{j=0}^{2^M-1} g(j) \psi_i(unpack(j, m)) 2^{L-M} \\
&= \tfrac{1}{2^M} \textstyle\sum_{j=0}^{2^M-1} g(j) \psi_i(unpack(j, m)) \\
&\qquad \text{we use Pack/Unpack Equivalency to complete} \\
&\qquad \text{the change of dimension} \\
&= \tfrac{1}{2^M} \textstyle\sum_{j=0}^{2^M-1} g(j) \psi_{pack(i,m)}(j) \\
&= w^g_{pack(i,m)}
\end{aligned}
$$

□

The Embedding Theorem means that the only indices for the nonzero Walsh coefficients of an embedding of a function occur when all of the one bits for the index fall entirely in the embedding mask. Therefore, if $f = \mathcal{E}(g, m)$, then there are at most $\binom{bc(m)}{k}$ Walsh coefficients w_i of f where $w_i \neq 0$ and $bc(i) = k$. Specifically, for a mask m such that $bc(m) = k$, there will be exactly one index for a nonzero Walsh coefficient of the expanded function with k one bits set and it will be the one that exactly matches the embedding mask. We will use this counting argument later to explore the general epistatic connectedness of functions.

Another way to view the statement of this theorem is that all the nonzero Walsh coefficients of $\mathcal{E}(g, m)$ lie within the hyperplane $h_{\overline{m},0}$ in the Walsh space. These are exactly the Walsh coefficients of the unembedded function g. This can best be illustrated in a similar matrix to the one we presented in equation 3.6.

$$
\begin{array}{c}
 \\
\uparrow \\
pack(x,\overline{m}) \\
\downarrow \\
 \\
 \\
\end{array}
\begin{array}{c}
\longrightarrow pack(x,m) \longrightarrow \\
\left[\begin{array}{ccccc}
w_0 & w_1 & w_2 & \ldots & w_{2^{bc(m)}-1} \\
0 & 0 & 0 & \ldots & 0 \\
\vdots & \vdots & \vdots & & \vdots \\
0 & 0 & 0 & \ldots & 0 \\
0 & 0 & 0 & \ldots & 0
\end{array}\right]
\end{array}
\tag{3.7}
$$

Notice the indexing of the Walsh coefficients is by value of $pack(x,m)$ and hence is in $\mathcal{B}^{bc(m)}$.

The Embedding Theorem unfortunately did not guarantee that there was a $1-1$ mapping between Walsh coefficients of g and its embedding. The following corollary gives us that assurance.

Corollary 31a

If $f = \mathcal{E}(g,m)$ then there is a 1-1 correspondence between $w^g_{pack(i,m)}$ and the potential nonzero Walsh coefficients of f, w^f_i.

Proof:

The proof is by showing there is a 1-1 correspondence between $i \in \mathcal{B}^L$ with $i \subseteq m$ and $j \in \mathcal{B}^M$ with $j = pack(i,m)$. It is clear from the Embedding Theorem that for every i such that $i \subseteq m$ there exists exactly one j that maps to it. Also the reverse function can be computed:

$$
\begin{aligned}
j &= pack(i,m) \\
unpack(j,m) &= unpack(pack(i,m),m) \\
unpack(j,m) &= i \wedge m \\
unpack(j,m) &= i \qquad \text{since } i \subseteq m
\end{aligned}
$$

Therefore each j maps to exactly one i. Therefore the mapping is a 1-1 correspondence.

□

The Embedding Theorem shows us that even though $\mathcal{E}(g, m)$ may be of considerably larger dimension than g, the computation of w_i^f is no harder. This is a hint of things to come. If a function is composed of much simpler subparts, the computation of the Walsh coefficients might be no more difficult than the computation of the Walsh coefficients of the subparts themselves.

The next theorem makes an important augmentation by allowing us to negate some or all of the bits selected by the embedding mask.

Theorem 32 (Embedding with Reflection)

Let $f : \mathcal{B}^L \rightarrow \mathbb{R}$ and $g : \mathcal{B}^M \rightarrow \mathbb{R}$ such that $f(x) = g(pack(x, m) \oplus n)$ where $m \in \mathcal{B}^L$, $bc(m) = M$, and $n \in \mathcal{B}^M$ then

$$w_i^f = \begin{cases} \psi_i(unpack(n, m)) w^g_{pack(i,m)} & \text{if} \quad i \subseteq m \\ 0 & \text{if} \quad i \not\subseteq m \end{cases}$$

Proof:

Let $h(x) = g(x \oplus n)$ where g and n are defined above then by the Reflection of Function Arguments Theorem:

$$w_j^h = \psi_j(n) w_j^g \qquad j \in \mathcal{B}^M$$

then we can embed h in $\mathcal{B}^L$ giving us $f(x) = h(pack(x, m))$. Therefore by the Embedding Theorem:

$$w_i^f = \begin{cases} w^h_{pack(i,m)} & \text{if} \quad i \subseteq m \\ 0 & \text{if} \quad i \not\subseteq m \end{cases}$$

where $i \in \mathcal{B}^L$. But $h(x) = g(x \ominus n)$ so $f(x) = g(pack(x, m) \ominus n)$ giving us:

$$w_i^f = \begin{cases} \psi_{pack(i,m)}(n) w^g_{pack(i,m)} & \text{if} \quad i \subseteq m \\ 0 & \text{if} \quad i \not\subseteq m \end{cases}$$

then by the Pack/Unpack Equivalency Theorem:

$$w_i^f = \begin{cases} \psi_i(unpack(n, m)) w^g_{pack(i,m)} & \text{if} \quad i \subseteq m \\ 0 & \text{if} \quad i \not\subseteq m \end{cases}$$

□

This theorem shows that, as with the Reflection of Function Arguments Theorem, the Walsh coefficients are at most altered by sign with the inclusion of the negation of bits selected in the mask. The sign is dependent only on the parity of the bits selected by the mask and the **negation mask**, n, itself.

The results of this theorem can be expressed, in a similar way to Equation 3.7, in the following matrix:

$$\begin{array}{c} \\ \\ pack(x, \overline{m}) \end{array} \begin{array}{c} \longrightarrow \; pack(x, m) \; \longrightarrow \\ \uparrow \\ \\ \downarrow \end{array} \begin{bmatrix} \psi_0(n)w_0 & \psi_1(n)w_1 & \psi_2(n)w_2 & \dots & \psi_{2^{bc(m)}-1}(n)w_{2^{bc(m)}-1} \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix} \tag{3.8}$$

Notice the indexing of the Walsh coefficients is by value of $pack(x, m)$ and hence is in $\mathcal{B}^{bc(m)}$.

3.11 Unembedding

For symmetry and completeness. I define the contraction of a function by a mask. This is essentially unembedding.

v is said to be a **contraction** of f if there is a mask $m : bc(m) = K$ and $m \in \mathcal{B}^L$ such that $v(x) = \langle f \rangle h_{m,x} \quad \forall x \in \mathcal{B}^K$. In this case m is the **contraction mask** and we denote v as a contraction of f using mask m by $\mathcal{C}(f, m) = v$.

In general. expansion is how to embed a lower dimension function in a higher dimension function by setting the fitness of every string in the hyperplane $h_{m,x}$ of f to $v(x)$. Contraction. on the other hand. is how to extract a function from a partition by setting the fitness of each string x to $\langle f \rangle h_{m,x}$. In the second case. information is destroyed and the original function cannot be constructed even if the contraction mask is known.

Observations:

- if $\mathcal{E}(v, m) = f$ then $\mathcal{C}(f, m) = v$
- if $\mathcal{C}(f, m) = v$ then it is **not** necessarily the case that $\mathcal{E}(v, m) = f$

3.12 Summary

We developed the concepts of hyperplanes and function embedding. This allows us to examine the values and epistasis of a function over small portions of its domain defined by subsets of bits. Embedding allows us to construct a function of larger domain out of a function of smaller domain. This essentially finishes our study of Walsh analysis in general.

In the next three chapters we will apply what we have learned: first to polynomials and decoding functions and then to logical expressions. and finally sums of embedded functions. The last case involves several familiar famous problems.

Chapter 4

Walsh Measures of Epistatic Structure

Walsh coefficients allow us to quantify epistasis for any subset of bits in the domain of a function. However, for a function of L bits, this yields 2^L coefficients which is too large a number to be practical. There are a variety of ways of organizing and reducing the dimensionality of this information to usefully represent the interactions between bits. I will present several schemes for counting nonzero Walsh coefficients, summing select subsets of coefficients, and computing the maximum number of bits of interaction.

Two major results of this chapter are the Walsh analysis of polynomial fitness functions and the Walsh analysis of various decodings, translations, and scalings of arguments. This work allows one to compute the the values of Walsh coefficients in these cases and the maximum number of bits of epistasis. We will find that the epistasis of polynomials is constrained by many factors which tends to create an upper limit on the number of bits of epistasis. This result suggests that fitness functions that are made of polynomials using common encoding techniques have low epistasis. This could be one of the reasons why many common optimization problems found in industry and nature may be solvable by evolutionary means.

Another result is that for certain classes of functions I find that the maximum epistasis may be reduced by properly selecting the range of the arguments to f_{model}. In an experiment, I will give an example of the application of the theory in the chapter. The epistasis of a problem will be reduced and a performance improvement will be seen. This begs the

question: will any kind of reduction of epistatic interaction result in an improvement in performance? I perform an experiment to see if some **patterns of epistasis** are better than others at changing the difficulty of the problem.

4.1 Walsh Coefficients and Epistasis

Chapter 2 showed that the Walsh transform was a linear transform and hence representable by a Walsh matrix. The Walsh matrix, Ψ, can be considered to be a row vector of column vectors. Thus the transform

$$\vec{f} = \Psi \vec{w}$$

becomes:

$$\vec{f} = [\ \psi_0 \quad \psi_1 \quad \ldots \quad \psi_{2^L-1}\ ]\,\vec{w}$$

where the ψ_i are the column vectors:

$$[\ \psi_i(0) \quad \psi_i(1) \quad \psi_i(2) \quad \ldots \quad \psi_i(2^L-1)\ ]^T$$

I also showed in chapter 2 that Ψ was orthogonal: hence Ψ represents an orthogonal basis. It was also noted that $\vec{w}$ was unique.

Recall the definition of ψ_i:

$$\psi_i(x) = Y^{\cdot}\Big(\bigoplus_{k=0}^{L-1}(x[k] \wedge i[k])\Big) \qquad x, i \in \mathcal{B}^L$$

From this definition, it is clear that *exactly* the bits contained in the subset of bits selected by bit string i, and *no others*, effect the value of ψ_i. That means each Walsh function's value is dependent on a different subset of bits and all possible subsets of bits are represented in the basis vectors found in Ψ. Therefore, each w_i uniquely represents the contribution of ψ_i to $\vec{f}$ and so uniquely represents the contribution of exactly the set of bits selected by i. So

a Walsh transform yields a transformation of coordinates to an orthogonal basis such that each w_i quantifies the epistasis between the subset of bits selected by i.

An examination of the Hyperplane Averaging Theorem reinforces this point. Consider Table 3.1. The average fitness for hyperplane *11 differs from the that for **1 by $-w_2$ and w_3. In order to account for the middle bit in hyperplane *11 being required to be 1, extra Walsh terms need to be added. w_2 represents the contribution to the function of the new bit in the middle position and w_3 the contribution of the interactions between rightmost two bits.

More formally, consider hyperplanes h and $\widehat{h}$, where $bc(\alpha(h)) - bc(\alpha(\widehat{h})) = 1$. Let $z = \alpha(h)) \ominus bc(\alpha(\widehat{h})$, that is, z is a mask that selects the bit that is not fixed in $\widehat{h}$ which is fixed in h. This means $\alpha(h) = z \ominus \alpha(\widehat{h})$. We know from the Hyperplane Averaging Theorem:

$$\begin{aligned}
\langle f \rangle_h &= \sum_{j : j \subseteq \alpha(h)} w_j \psi_j(\beta(h)) \\
&= \sum_{j : j \subseteq (z \ominus \alpha(\widehat{h}))} w_j \psi_j(\beta(h)) \\
&\qquad \text{The sum on the right can be broken down into two parts. The part with the bit set and the part with the bit not set.} \\
&= \sum_{j : j \subseteq \alpha(\widehat{h})} w_{z \ominus j} \psi_{z \ominus j}(\beta(h)) + \sum_{j : j \subseteq \alpha(\widehat{h})} w_j \psi_j(\beta(h)) \\
&\qquad \text{Since } j \text{ in the second sum is limited, } j \subseteq \alpha(\widehat{h}), \text{ the argument to the Walsh function can be similarly limited.} \\
&= \sum_{j : j \subseteq \alpha(\widehat{h})} w_{z \ominus j} \psi_{z \ominus j}(\beta(h)) + \sum_{j : j \subseteq \alpha(\widehat{h})} w_j \psi_j(\beta(\widehat{h})) \\
&= \sum_{j : j \subseteq \alpha(\widehat{h})} w_{z \ominus j} \psi_{z \ominus j}(\beta(h)) + \langle f \rangle_{\widehat{h}}
\end{aligned}$$

The last line shows that for each additional bit position in a hyperplane that is fixed a new set of Walsh coefficients must be added in. As can be seen in the first sum of the last line, those Walsh coefficients represent the interactions between the newly fixed bit position and all of the fixed bit positions in the previous hyperplane. Thus, as each bit position in a hyperplane becomes fixed the Walsh coefficients for that bit position must be included. In the extreme case all bit positions are added in and all Walsh coefficients are included. This

shows that the Walsh coefficients do truly represent bit interactions.

4.2 Epistatic Measures

A complete set of Walsh coefficients is an array of reals as large as the original function table. It is convenient to group the Walsh coefficients into a more manageable and meaningful set of measures. There are two obvious approaches to this. The first approach is to capture the structure of the interactions. Each **nonzero Walsh coefficient nonzero Walsh coefficients** indicates the existence of a particular bit interaction. A Walsh coefficient that is **zero** means the subset of bits, taken as a whole, is independent. Therefore, any interaction between bits in the subset must happen between smaller subsets of bits. For example: if $w_7 = 0$, then the lower 3 bits do not interact epistatically, but the lower two bits might interact, i.e. $w_3 \neq 0$. In fact, any or all of the 3 pairs of bits, 3 singleton bits, and no bits (w_0) may interact epistatically and w_7 remain zero. Therefore, measures that record just the number and/or distribution of nonzero Walsh coefficients record the structure of the interactions but not the magnitudes. The second approach is to capture the magnitudes in some way. In the following sections are several possible measures that I will use throughout the remainder of the dissertation.

4.2.1 Coverage

One of the simplest measures of structure is **coverage** which is the ratio of the number of nonzero Walsh coefficients to the total number of Walsh coefficients. This gives a value between 0 and 1 representing the number of *contributing*, nonzero, bit interactions over the total number available.

4.2.2 The Walsh Count

Another measure for epistatic structure is the **Walsh count**. It measures the diversity of interaction for a given number of interacting bits. It is denoted by $\mathcal{K}_b$ where:

$$\mathcal{K}_b = \sum_{i:bc(i)=b} NZ(w_i)$$

with NZ returning 1 if w_i is nonzero and 0 otherwise, and $bc(i)$ returning the bit count of i. $\mathcal{K}_b$ is, therefore, the number of nonzero Walsh coefficients for bit patterns i with exactly b bits set to 1. Notice that for an L bit function, there are $L+1$ Walsh counts. The vector $\vec{\mathcal{K}}$ is the distribution of nonzero Walsh coefficients. I call this the **Walsh distribution**. The maximum value of $\mathcal{K}_b$ is $\binom{L}{b}$, and therefore for a random function, the values in $\vec{\mathcal{K}}$ form a binomial distribution.

4.2.3 Function Order

Let the **order of a function**, denoted $\Omega(f)$, be defined as the largest i such that $\mathcal{K}_i \neq 0$. In the special case where all $\mathcal{K}_i$ are 0, that is $f(x) = 0$, $\Omega(f) = 0$.

Observations:

- If $\Omega(f) = 0$ then $f(x) = c$.
- If $\Omega(f) = 1$ then the function is **linear** and there are no pairwise or higher order interactions amongst the bits in the domain.
- $\Omega(S_i) = 0$ or i where S_i is the i^{th} spectral function. ($\Omega(S_i) = 0$ iff $S_i = 0$)

I will prove the first two of these observations later in this chapter.

4.2.4 Walsh Sums

An epistatic measure that measures the magnitude of epistasis the **Walsh sum**. It is denoted by W_b where:

$$W_b = \sum_{i:bc(i)=b} |w_i|$$

and $bc(i)$ is the bit count of i. That is W_b is the sum of the **absolute value** of all of the Walsh coefficients for bit patterns i with exactly b bits set to 1. W_b is the b^{th} **order Walsh sum**.

For example:

$$
\begin{aligned}
W_0 &= |w_0| \\
W_1 &= |w_1| + |w_2| + |w_4| + |w_8| + \ldots + |w_{2^{(L-1)}}| \\
W_2 &= |w_3| + |w_5| + |w_6| + |w_9| + \ldots + |w_{3*2^{(L-2)}}| \\
&\vdots \\
W_L &= |w_{2^L-1}|
\end{aligned}
$$

Observations:

- For an L bit function. there are $L + 1$ Walsh sums.
- W_k is the sum of the absolute value of the $\binom{L}{k}$ Walsh coefficients.
- Since the absolute value of the Walsh coefficients is used. Walsh coefficients never cancel and Walsh sums are always nonnegative. This means any nonzero Walsh coefficient will force the containing Walsh Sum to be nonzero. Therefore. Walsh Sums measure the magnitude of presence of nonzero Walsh coefficients and hence the level of n-bit interactions.

The following theorem makes an important point about Walsh sums.

Theorem 33 (Walsh Sum Representation)

W_j is the sum of the absolute values of the w_i that represent j bit interactions.

Proof: The various w_i used to compute W_j are exactly the w_i that occur in the Hyperplane Averaging Theorem to compute the hyperplane averages for order j hyperplanes. but not any lower order hyperplanes. Therefore. the w_i used to compute W_j are the Walsh coefficients sufficient to express the increased order of interaction in order j hyperplanes over order $j-1$ hyperplanes. From this. we conclude that W_j is a measure of the magnitude of order j bit interactions: i.e.. it measures the effects on the function value that can be attributed *solely* to j bit interactions.

There is a close relationship between spectral functions and Walsh sums. The only possible nonzero Walsh sum for S_i is W_i and that W_i for S_i from the spectral decomposition of f is the same as the W_i for f itself. Spectral functions enable us to more easily express ideas about functions with limited degrees of interaction. In particular, since spectral decomposition isolates Walsh coefficients according to parity, it is also a useful way to decompose functions that display parity.

Observations:

- $W_i \neq 0$ iff $\exists j \ : \ bc(j) = i$ and $w_i \neq 0$
- The only possible nonzero Walsh sum for S_i is W_i.
- $S_0 = w_0$

Intuitively, the order of a function is the size of the largest set of interdependent bits. So $\Omega(f)$ is a measure of the level of epistasis of f. W_i measures the magnitude of the i-bit interdependence, and $\mathcal{K}_i$ measures the number of i-bit interactions.

4.3 The Epistatic Structure of Polynomials

Given the mathematical expression for function f, what is known about $\Omega(f)$? In this section, I develop a set of theorems that aid in predicting bit interaction based on the mathematical expression for the function.

Theorem 34 (Ω of a Constant)

If $f(x) = c$, then $\Omega(f) = 0$

Proof:

If $f(x) = c$, then by the Balanced Sum Theorem (See Appendix) only w_0 can be nonzero. If $c = 0$ then this is the special case of $\Omega(f) = 0$. Therefore $\Omega(f) = 0$.

Observations:

- if $f(x) = c$ then $w_k = \begin{cases} c & \text{if } k = 0 \\ 0 & \text{otherwise} \end{cases}$

Theorem 35 (Ω of x)

If L bit function $f(x) = x$ then $\Omega(f) = 1$

Proof:

It is clear that $w_0 = (2^L - 1)/2$ is the average of the numbers from 0 to $2^L - 1$. By setting $w_{2^k} = -2^{k-2}$ for $k = 1$ to $L - 1$ we get the Walsh coefficients based on the place value worth of each binary bit position divided by 2. It is obvious from this scheme that all other Walsh coefficients must be zero. In summary, for $f(x) = x$:

$$w_k = \begin{cases} (2^L - 1)/2 & \text{if } k = 0 \\ -2^{t-1} & \text{if } k = 2^t \\ 0 & \text{otherwise} \end{cases}$$

Since Walsh coefficient solutions are unique, this solution is unique. (One proof of uniqueness is based on the observation that the Walsh coefficients and function values can be treated as two vectors related by a multipling by a matrix of Walsh functions. That matrix is singular, therefore there is a unique inverse.) Notice that the only nonzero Walsh coefficients are for strings with $bc(j) \leq 1$, therefore $\Omega(f) = 1$.

A constructive version of the proof is not as intuitive but reveals more about the origin of epistasis. The proof proceeds methodically as follows:

Proof:

This proof is done by establishing values for the Walsh coefficients of an L bit function f based on the $L - 1$ bit version function of f. Let w_j^k denote the j^{th} Walsh coefficient for function f over a k bit domain. Then we know that for function f in the L bit domain:

$$
\begin{aligned}
w_j^L &= \tfrac{1}{2^L}\textstyle\sum_{x=0}^{2^L-1} x\psi_j(x) \\
&= \tfrac{1}{2^L}\textstyle\sum_{x=0}^{2^{L-1}-1} x\psi_j(x) + \tfrac{1}{2^L}\sum_{x=2^{L-1}}^{2^L-1} x\psi_j(x) \\
&= \tfrac{1}{2^L}\textstyle\sum_{x=0}^{2^{L-1}-1} x\psi_j(x) + \tfrac{1}{2^L}\sum_{x=0}^{2^{L-1}-1}(x+2^{L-1})\psi_j(x+2^{L-1}) \\
&= \tfrac{1}{2^L}\textstyle\sum_{x=0}^{2^{L-1}-1} x\psi_j(x) + \tfrac{1}{2^L}\sum_{x=0}^{2^{L-1}-1} x\psi_j(x+2^{L-1}) + \tfrac{1}{2^L}\sum_{x=0}^{2^{L-1}-1} 2^{L-1}\psi_j(x+2^{L-1}) \\
&= \tfrac{1}{2^L}\textstyle\sum_{x=0}^{2^{L-1}-1} x(\psi_j(x)+\psi_j(x+2^{L-1})) + \tfrac{1}{2}\sum_{x=0}^{2^{L-1}-1}\psi_j(x+2^{L-1})
\end{aligned}
$$

This results in four cases based on two observations. The first observation is by the definition of ψ over an L bit domain:

$$
\psi_j(x+2^{L-1}) = \begin{cases} \psi_j(x) & \text{if } j < 2^{L-1} \\ -\psi_j(x) & \text{if } j \geq 2^{L-1} \end{cases} \qquad \text{for } x \in \mathcal{B}^{L-1}, j \in \mathcal{B}^L
$$

The change in sign is caused by the inclusion of the L^{th} bit in the parity check in ψ.

The second observation is based on the first and the Balanced Sum Theorem:

$$
\sum_{x=0}^{2^{L-1}-1} \psi_j(x+2^{L-1}) = \begin{cases} 2^{L-1} & \text{if } j = 0 \\ -2^{L-1} & \text{if } j = 2^{L-1} \\ 0 & \text{otherwise} \end{cases}
$$

Picking up where we left off: the observations induce four different cases which I present in parallel.

$$w_j^L = \begin{cases} \frac{1}{2^L}\sum_{x=0}^{2^{L-1}-1} x(\psi_j(x)+\psi_j(x)) + 2^{L-1}/2 & \text{if } j=0 \\ \frac{1}{2^L}\sum_{x=0}^{2^{L-1}-1} x(\psi_j(x)+\psi_j(x)) & \text{if } 0<j<2^{L-1} \\ \frac{1}{2^L}\sum_{x=0}^{2^{L-1}-1} x(\psi_j(x)-\psi_j(x)) - 2^{L-1}/2 & \text{if } j=2^{L-1} \\ \frac{1}{2^L}\sum_{x=0}^{2^{L-1}-1} x(\psi_j(x)-\psi_j(x)) & \text{if } j>2^{L-1} \end{cases}$$

$$= \begin{cases} \frac{1}{2^{L-1}}\sum_{x=0}^{2^{L-1}-1} x\psi_j(x) + 2^{L-2} & \text{if } j=0 \\ \frac{1}{2^{L-1}}\sum_{x=0}^{2^{L-1}-1} x\psi_j(x) & \text{if } 0<j<2^{L-1} \\ -2^{L-2} & \text{if } j=2^{L-1} \\ 0 & \text{if } j>2^{L-1} \end{cases}$$

$$= \begin{cases} w_j^{L-1} + 2^{L-1} & \text{if } j=0 \\ w_j^{L-1} & \text{if } 0<j<2^{L-1} \\ -2^{L-2} & \text{if } j=2^{L-1} \\ 0 & \text{if } j>2^{L-1} \end{cases}$$

Hand calculation reveals that for $L = 1$: $w_0^1 = \frac{1}{2}$ and $w_1^1 = -\frac{1}{2}$. Solving the recurrence above gives:

$$w_j^L = \begin{cases} (2^L - 1)/2 & \text{if } j=0 \\ -2^{L-1} & \text{if } j=2^i \\ 0 & \text{otherwise} \end{cases}$$

Notice that the only nonzero Walsh coefficients are for strings with $bc(j) \leq 1$. therefore $\Omega(f) = 1$.

The next theorem shows how the scalers affect the value of Ω.

Theorem 36 (Linear Composition)

If C is a nonzero constant and D is any constant.

$$\Omega(Cf + D) = \Omega(f)$$

Proof:

Let

$$\begin{aligned} g(x) = Cf(x) + D &= C(\textstyle\sum_{i=0}^{2^L-1} w_i\psi_i(x)) + D \\ &= (\textstyle\sum_{i=0}^{2^L-1}(Cw_i)\psi_i(x)) + D \\ &= \textstyle\sum_{i=1}^{2^L-1}(Cw_i)\psi_i(x) + (Cw_0 + D)\psi_0(x) \end{aligned}$$

Denote the k^{th} Walsh coefficient of g above as w_k^g. Since by the definition of $\Omega(f)$, if $bc(k) > \Omega(f)$ then $w_k = 0$ we know that $w_k^g = Cw_k = 0$ for the same k.

The next two theorems consider the merging of functions with binary operators.

Theorem 37 (Sum of Functions)

For any two L bit functions f and g

$$\Omega(f + g) \leq \max(\Omega(f), \Omega(g))$$

Proof:

Let w^f be the Walsh coefficients for the function f and similarly for g. Then

$$f(x) = \textstyle\sum_{i=0}^{2^L-1} w_i^f\psi_i(x) \quad \text{and} \quad g(x) = \textstyle\sum_{i=0}^{2^L-1} w_i^g\psi_i(x)$$

Therefore

$$\begin{aligned} f(x)+g(x) &= \sum_{i=0}^{2^L-1} w_i^f \psi_i(x) + \sum_{i=0}^{2^L-1} w_i^g \psi_i(x) \\ &= \sum_{i=0}^{2^L-1} (w_i^f + w_i^g)\psi_i(x) \\ &= \sum_{i=0}^{2^L-1} w_i^{f-g}\psi_i(x) \end{aligned}$$

where w^{f-g} denotes Walsh coefficients of $f+g$. From the above it is clear that

$$w_i^{f-g} = w_i^f + w_i^g$$

Since both w_i^f and w_i^g are zero if $bc(i) > \max(\Omega(f), \Omega(g))$ then $(w_i^f + w_i^g) = 0$ as well. Therefore

$$\Omega(f+g) \leq \max(\Omega(f), \Omega(g))$$

□

I use the Product Coefficient Theorem from the Chapter 2 to put constraints on Ω for products of functions and hence polynomials, as we see in the next theorem.

Theorem 38 (Product of Functions)

For the product of any two L bit functions f and g

$$\Omega(fg) \leq \Omega(f) + \Omega(g)$$

Proof:

From the Product Coefficient Theorem, we know:

$$w_k^{fg} = \sum_{j=0}^{2^L-1} w_j^f w_{k \div j}^g$$

Therefore, w_k^{fg} can only be nonzero when both $w_j^f \neq 0$ and $w_{k \div j}^g \neq 0$ for some value of j. Since $bc(j) \leq \Omega(f)$ and $bc(k \oplus j) \leq \Omega(g)$, $bc(k)$ can never have more than $\Omega(f) + \Omega(g)$ bits.

The previous theorems leads us logically to combine them in a general theorem covering any polynomial with nonnegative exponents.

Theorem 39 (Polynomial Complexity)

Let P_n be a polynomial with degree $n : n \geq 0$ such that $P_n : \mathcal{B}^L \rightarrow \mathbb{R}$ then

$$\Omega(P_n) \leq n$$

Proof:

Consider x^n. Since $\Omega(x) = 1$ and by the Product Theorem $\Omega(f^k) \leq k\Omega(f)$, we find $\Omega(x^n) \leq n\Omega(x) = n$. Therefore, by the Linear Composition Theorem, a term of $P_n : ax^n$ has $\Omega(ax^n) \leq n$. Hence if $P_n = \sum_{k=0}^{n} a_k x^k$: $a_n \neq 0$, then

$$\Omega(P_n) \leq \max_{k=0..n} \Omega(a_k x^k) = n: \quad a_n \neq 0, k \geq 0$$

4.4 Parity of Functions

A close examination of the convolution in the Product Coefficient Theorem suggests that functions may have a parity property with respect to the Walsh sums and spectral functions.

Theorem 40 (Function Parity)

Let f and g be two functions with $W_{n_f}^f$ being the only nonzero Walsh sum for f and $W_{n_g}^g$ being the only nonzero Walsh sum for g, then the product of two functions, fg, can only have nonzero Walsh sums at $W_{n_f+n_g-2m}^{fg}$ for m, a nonnegative integer.

Proof:

From the Product Coefficient Theorem we know that

$$w_k^{fg} = \sum_{j=0}^{2^L-1} w_j^f w_{k \dot{-} j}^g$$

Therefore, $w_{k'}^{fg}$ can only be nonzero when both $w_j^f \neq 0$ and $w_{k' \ominus j}^g \neq 0$ for some value of j. This, by our premise, occurs when $bc(j) = n_f$ and $bc(k' \ominus j) = n_g$. It is easy to see that the allowable values for $bc(k')$ that fit these constraints cannot have more that $n_g + n_f$ bits. In fact mutual cancellation of bits in $k' \ominus j$ means that $bc(k') = n_f + n_g - 2m$.

Another way to look at this is to make a change of variable for any given j as follows. Let $k = j \ominus k'$. Then our requirement for nonzeroness

$$w_{k'}^{fg} \neq 0 \quad \text{if} \quad bc(j) = n_f \quad \text{and} \quad bc(k' \ominus j) = n_g$$

becomes

$$w_{k \ominus j}^{fg} \neq 0 \quad \text{if} \quad bc(j) = n_f \quad \text{and} \quad bc(k) = n_g$$

Notice that the index of the Walsh coefficient is the exclusive-or of a n_f bit string and a n_g bit string which must be a $n_f + n_g - 2m$ bit string where m is a nonnegative integer. Therefore the only nonzero Walsh sums are either all odd numbered or all even numbered and the theorem is proved.

□

The Function Parity Theorem implies important parity properties for functions. Let's begin with some definitions. A function which has only even order nonzero Walsh sums is called an **even order function**. A function which has only odd order nonzero Walsh sums is called **odd order function**. This property is called **function parity**.

Classically, f is an **even function**, as opposed to an *even order function*, if and only if $f(x) = f(-x)\ \forall x \in \mathbb{R}$. An **odd function**, similarly, has the property that $f(x) = -f(-x)$. A similar relation holds for even order and odd order functions as we see in the next theorem.

Theorem 41 (Function Parity)

If $\overline{x}$ represents the logical negation of the bits of x:

A function f is an even order function if and only if it has the property that $f(x) = f(\overline{x})$.

A function f is an odd order function if and only if it has the property that $f(x) = -f(\overline{x})$.

Proof:

Let $\overline{f}$ denote $f(\overline{x})$. Let's look at the relationship between the Walsh coefficients for f and $\overline{f}$. Note that $f(\overline{x}) = f(x \ominus \vec{1})$ which is the reflection of f about the point $\vec{1}$. Using the Reflection of Function Arguments Theorem we see that:

$$\begin{aligned} w_i^{\overline{f}} &= \psi_i(\vec{1}) w_i^f \\ &= (-1)^{bc(i)} w_i^f \end{aligned}$$

If $f = \overline{f}$, then the Walsh coefficients of the functions must be equal and so it is immediately clear that $bc(i)$ must be even for any $w_i \neq 0$. Hence, the w_i are zero where $bc(i)$ is odd. It is also clear from the equation that if f is an odd order function then $f = \overline{f}$. Hence the first half of the theorem is proven.

The second half follows by the same logic.

The Function Parity Theorem yields some useful **parity laws**.

It can't be emphasized enough that these parity properties apply to functions over the finite range $\mathcal{B}^L$. For example, it will be shown later that when the domain is first mapped by specific scaling and offset, cosine, $\cos : \mathcal{B}^L \rightarrow \mathcal{R}$, is an even order function. This does **not** mean that $\cos : \mathcal{R} \rightarrow \mathcal{R}$ is an even order function.

Theorem 42 (Parity Laws)

1. The product of an odd order function with an even order function is an odd order function.
2. The product of two odd or two even order functions is an even order function.

Proof:

Consider, first, the product of two even order functions, f and g. Since the functions are even order and any function in $\mathcal{B}^L$ can be decomposed into the sum of spectral functions:

$$f(x) = \sum_{j=0}^{\lfloor L/2 \rfloor} S_{2j}^f(x) \quad \text{and} \quad g(x) = \sum_{k=0}^{\lfloor L/2 \rfloor} S_{2k}^g(x)$$

then

$$fg(x) = \sum_{j=0}^{L/2} \sum_{k=0}^{L/2} S_{2j}^{f}(x) S_{2k}^{g}(x) \tag{4.1}$$

From the Function Parity Theorem it is immediately evident that the product of any two even spectral functions S_j, S_k results in a function that is the sum of a series of even order spectral functions:

$$S_j S_k = S_0^{jk} + S_2^{jk} + \cdots + S_{j-k}^{jk}$$

Therefore, each product of S_j and S_k in the Equation 4.1 can be replaced by a sum of even order spectral functions S_{2i}^{jk} as follows:

$$fg(x) = \sum_{j=0}^{L/2} \sum_{k=0}^{L/2} \sum_{i=0}^{j-k} S_{2i}^{jk}(x)$$

and hence fg is an even order function. Similar arguments hold for the other two cases.

□

It turns out that by cleverly selecting the domain for a function, some functions can be forced to be even or odd order. I call this technique **argument centering**. It is based on the observation: if a domain of a function is itself a linear function over $\mathcal{B}^L$ then it has an Ω of 1. So it must be the sum of the two spectral functions S_0 and S_1. But S_0 is just the mean of all of the function values. By adjusting the mean of the linear function, which is the domain, to zero, I can force $S_0 = 0$. This leaves the function equal to the single spectral function S_1 which is an odd order function. If a polynomial is applied to an argument that is an odd order function then the Parity Laws apply throughout any products occurring in the polynomial. The result is that for a polynomial in x, odd powers of x are odd order functions and even powers of x are even order functions. So if the polynomial is all even powers or all odd powers, the parity of the polynomial can be predicted.

As an example, consider the function $f(z) = \sin(z)$ for $z \in \mathcal{B}^L$. Each of the 2^L values of the function can be computed using the Taylor series expansion about 0 for sin:

$$\sin(x) = x/1! + x^3/3! - x^5/5! + ...$$

$z \in [0.2^L - 1]$ can be thought of as a linear function $g(z) : \mathcal{B}^L \rightarrow \mathbb{R}$ and function $f(z) = \sin(g(z))$. In this example, g is simply the identity mapping of binary to reals. Since g is linear it can be represented as a sum of two spectral functions:

$$g = S_0 + S_1$$

By definition S_0 is the average of the values of g which is $(2^L - 1)/2$. I can now use g to alter the domain of f. Let $g' = g - S_0$ and $f'(z) = sin(g'(z))$ for $z \in \mathcal{B}^L$ then:

$$\begin{aligned} g' &= g - S_0 \\ &= (S_0 + S_1) - S_0 \\ &= S_1 \end{aligned}$$

Now f is defined over domain $g'(z)$ for $z \in \mathcal{B}^L$, that is, $g'(x) \in [-(2^L - 1)/2, (2^L - 1)/2]$. The domain of f now appears centered, since the average of the values of the domain is zero and the function is linear. Since $g' = S_1$, g' is an odd order function. The parity laws now apply. Hence, each term in the Taylor series is now an odd order function and so is f'. This means that the even order Walsh coefficients must be zero.

Argument centering is somewhat of a misnomer. Note that it is only necessary that the domain be linear and the average of the domain values be 0 in order for argument centering to force the domain to be an odd order function. That is, actually centering the domain is not required. We will see several more examples of argument centering and discuss the advantages and disadvantages of altering the domain when I "test drive" the theorems in the empirical section of this chapter.

The previous sections show that we should be able to predict the Walsh coefficients and maximum number of bits of epistasis for a simple polynomial. In the next section I

build some tools for working with f_{decode} and put it all together in a general theorem for predicting epistasis.

4.5 The Walsh Structure of Parameter Decoding

Parameter decoding is a very important part of preparing a problem for solution with an evolutionary algorithm (Mitchell et al., 1992; Mathias and Whitley, 1994; Whitley et al., 1995b). From a practical standpoint, parameter decoding consists of three steps. First, the subset of bits that encode each parameter are extracted from the bitstring. The substrings are then decoded into integers and finally the integers scaled to the appropriate range for each parameter of f_{model}. The first two steps are usually performed using logical operators on the bitstring representing the chromosome. The last step can be incorporated as part of the model function.

In this section I construct some theorems to account for the bit interactions introduced by parameter encoding. I begin with a simple observation about reflecting the argument by the EXCLUSIVE-OR of a constant. I then proceed to show the effects of extraction and Gray coding.

4.5.1 Reflection

As we saw earlier, if g represents the function f reflected about r, then f can be defined as $g(x) = f(x \oplus r)$. It turns out that Walsh sums are invariant under reflection as we see in this next theorem.

Theorem 43 (Walsh Sums under Reflection)

Reflection of a function about any number of axes has no effect on the Walsh sums of the function.

Proof:

Let g be a reflection of function f about z then by the Reflection of Function Arguments Theorem:

$$w_i^g = \psi_i(z) w_i^f$$

This is only a sign change for the Walsh coefficients. and since Walsh sums use the absolute value of Walsh coefficients. the sign is irrelevant.

□

From this it follows immediately that

Corollary 43a

$$\Omega(f(x)) = \Omega(f(x \ominus z)) \text{ for any } z$$

These two theorems show that both Walsh sum and Ω measures are invariant under reflection. This is a side effect of the dimensionality reduction in deriving the measures. If algorithm performance varies under reflection of the problem. then this would be a weakness of these measures.

4.5.2 Extraction

Extraction is often the basis of composing parameters for functions with domain $\mathbb{R}^n$ from strings in $\mathcal{B}^L$ that are processed by genetic algorithms. Let $x[n_1, n_2]: \mathcal{B}^L \times \mathbb{Z}_L \times \mathbb{Z}_L \rightarrow \mathcal{B}^L$ be the **extraction operator** which extracts bits in positions n_1 through n_2 ($n_2 \geq n_1$) from string x and places them in the least significant bit portion of the string with zero fill. This operation can be performed by masking and shifting. This leads us to our next general theorem which can be used to compute the Ω for a variety of logical operators.

Let the operator $\xrightarrow{\text{shift}}: \mathcal{B}^L \times \mathbb{Z}_L \rightarrow \mathcal{B}^L$ be the binary right shift operator applied to left operand. shifting it by the number of bits found in the right operand. The result is zero filled from the left. $\xleftarrow{\text{shift}}$ is defined similarly. For example: if $s = 01010111$ then $(s \xrightarrow{\text{shift}} 2) = 01011100$ and $(s \xrightarrow{\text{shift}} 3) = 00001010$. To prove the theorems on extraction. I will actually use a generalization of extraction based on masking and shifting.

Theorem 44 (General Extraction)

$$\Omega(f((x \wedge m) \xrightarrow{\text{shift}} s)) \leq \min(\Omega(f(x)). bc(m). L - s)$$

Where m is a mask and s is the amount to shift the result of $x \wedge m$.

Proof:

$$
\begin{aligned}
f((x \wedge m) \xrightarrow{\text{shift}} s) &= \textstyle\sum_{i=0}^{2^L-1} w_i \psi_i((x \wedge m) \xrightarrow{\text{shift}} s) \\
&= \textstyle\sum_{i=0}^{2^L-1} w_i Y(\bigoplus_{j=0}^{L-1}((x \wedge m) \xrightarrow{\text{shift}} s)[j] \wedge i[j]) \\
&= \textstyle\sum_{i=0}^{2^L-1} w_i Y(\bigoplus_{j=0}^{L-s-1}((x \wedge m) \xrightarrow{\text{shift}} s)[j] \wedge i[j]) \\
&= \textstyle\sum_{i=0}^{2^L-1} w_i Y(\bigoplus_{j=0}^{L-s-1}((x \wedge m)[j+s] \wedge i[j])) \\
&= \textstyle\sum_{i=0}^{2^L-1} w_i Y(\bigoplus_{j=s}^{L-1}((x \wedge m)[j] \wedge i[j-s])) \\
&= \textstyle\sum_{i=0}^{2^L-1} w_i Y(\bigoplus_{j=0}^{L-1}((x \wedge m)[j] \wedge (i \xleftarrow{\text{shift}} s)[j])) \\
&\quad \text{note the reversal of the direction of shifting} \\
&= \textstyle\sum_{i=0}^{2^L-1} w_i Y(\bigoplus_{j=0}^{L-1}(x[j] \wedge m[j] \wedge (i \xleftarrow{\text{shift}} s)[j])) \\
&= \textstyle\sum_{i=0}^{2^L-1} w_i Y(\bigoplus_{j=0}^{L-1}(x[j] \wedge (m \wedge (i \xleftarrow{\text{shift}} s))[j])) \\
&= \textstyle\sum_{i=0}^{2^L-1} w_i \psi_{(m \wedge (i \xleftarrow{\text{shift}} s))}(x)
\end{aligned}
$$

We can see the contraction of the coefficient space from "all of i" to $(m \wedge (i \xleftarrow{\text{shift}} s))$. Since the coefficient for the Walsh function $\psi_i(x)$ is by definition the i^{th} Walsh coefficient, we can use the last expression above to compute the Walsh coefficient w'_i for the function $f((x \wedge m) \xrightarrow{\text{shift}} s)$ based on the Walsh coefficients w_i for $f(x)$.

$$
w'_i = \sum_{k \,:\, i=(m \wedge (k \xleftarrow{\text{shift}} s))} w_k
$$

This mapping of Walsh coefficients constrains the subscript j for **nonzero** Walsh coefficients in the following ways:

- $bc(j) \leq \Omega(f)$ since $(m \wedge (i \xleftarrow{\text{shift}} s))$ can only reduce the bit count for any nonzero Walsh coefficient.
- $bc(j) \leq bc(m)$ since m masks the whole expression.

- $bc(j) \leq L - s$ since the zero fill left shift further reduces the possible number of ones in j.

Therefore

$$\Omega(f((x \wedge m) \xrightarrow{\text{shift}} s) \leq \min(\Omega(f(x)),\ bc(m),\ L - s)$$

and the theorem follows.

□

The above theorem is more general than necessary for use with bit extraction.

Corollary 44a (Extraction Corollary)

$$\Omega(f(x[n_1, n_2])) \leq \min(\Omega(f(x)), bc(m))$$

Proof:

Since the extraction operator can be defined as:

$$x[n_1, n_2] = (x \wedge m) \xrightarrow{\text{shift}} s$$

the above theorem applies to the extraction operator when $m = \vec{1}_{n_1, n_2}$, which is a mask with bits n_1 through n_2 set to one, and $s = n_1$ (remembering that bits are numbered from 0). In the bit extraction case we see:

$$\begin{aligned} L &\geq n_2 + 1 \\ L - n_1 &\geq n_2 - n_1 + 1 \\ L - s &\geq bc(m) \end{aligned}$$

giving us:

$$\Omega(f(x[n_1, n_2])) \leq \min(\Omega(f(x)), bc(m))$$

□

Furthermore, by choosing the mask and shift values appropriately, we see that:

$$\Omega(f(x \xrightarrow{\text{shift}} s)) \leq \min(\Omega(f(x)), L - s)$$

and

$$\Omega(f(x \wedge m)) \leq \min(\Omega(f(x)), bc(m))$$

These relations are based on a common theme of finding a function $R : \mathcal{B}^L \rightarrow \mathcal{B}^L$ such that there is a function $R' : \mathcal{B}^L \rightarrow \mathcal{B}^L$ for which:

$$\psi_i(R(x)) = \psi_{R'(x,i)}(x) \qquad \forall x \in \mathcal{B}^L$$

As we have seen, this is certainly the case with logical function $R(x) = (x \wedge m) \xrightarrow{\text{shift}} s$ in which case $R'(x) = m \wedge (i \xrightarrow{\text{shift}} s)$. The following theorem about combining logical functions of this form will be used when dealing with Gray codes in the next section.

Theorem 45 (Logical Function Exclusive-or)

Given two functions $R, Q : \mathcal{B}^L \rightarrow \mathcal{B}^L$ and:

$$\psi_i(R(x)) = \psi_{R'(x,i)}(x) \quad \text{and} \quad \psi_i(Q(x)) = \psi_{Q'(x,i)}(x) \qquad \forall x \in \mathcal{B}^L$$

then

$$\psi_i(R(x) \ominus Q(x)) = \psi_{R'(x) \doteq Q'(x,i)}(x) \qquad \forall x \in \mathcal{B}^L$$

Proof:

$$\begin{aligned}\psi_i(R(x) \ominus Q(x)) &= \psi_i(R(x))\psi_i(Q(x)) \\ &= \psi_{R'(x,i)}(x)\psi_{Q'(x,i)}(x) \\ &= \psi_{R'(x,i) \dot{-} Q'(x,i)}(x)\end{aligned}$$

4.5.3 Gray Coding

I can apply the Logical Function Exclusive-or Theorem to the use of Gray codes. Consider the common encoding technique which is to assume the extracted bits are in a binary reflected Gray code. This Gray code is usually defined as:

$$gray(n) = n \ominus (n \xrightarrow{\text{shift}} 1)$$

If a value has been encoded with a Gray code then a **degray** function $degray : \mathcal{B}^L \rightarrow \mathcal{B}^L$ must be applied to convert the extracted argument to the desired value. A mathematically simple approach to degraying is through the series:

$$degray(n) = n \ominus (n \xrightarrow{\text{shift}} 1) \ominus (n \xrightarrow{\text{shift}} 2) \ominus \ldots$$

It is clear that this is the inverse of *gray* since:

$$\begin{aligned}degray(gray(n)) &= [n \ominus (n \xrightarrow{\text{shift}} 1)] \ominus ([n \ominus (n \xrightarrow{\text{shift}} 1)] \xrightarrow{\text{shift}} 1) \ominus ([n \ominus (n \xrightarrow{\text{shift}} 1)] \xrightarrow{\text{shift}} 2) \ominus \ldots \\ &= [n \ominus (n \xrightarrow{\text{shift}} 1)] \ominus [(n \xrightarrow{\text{shift}} 1) \ominus (n \xrightarrow{\text{shift}} 2)] \ominus [(n \xrightarrow{\text{shift}} 2) \ominus (n \xrightarrow{\text{shift}} 3)] \ominus \ldots \\ &= n \ominus [(n \xrightarrow{\text{shift}} 1)] \ominus (n \xrightarrow{\text{shift}} 1)] \ominus [(n \xrightarrow{\text{shift}} 2)] \ominus [(n \xrightarrow{\text{shift}} 2)] \ominus \ldots \\ &= n\end{aligned}$$

In practice, n would be an extraction of bits from x performed by $(x \wedge m) \xrightarrow{\text{shift}} s$, where x is the whole chromosome. Combining extraction with the degray function we get:

$$degray(x[i.j]) = ((x \wedge m) \xrightarrow{\text{shift}} s) \ominus ((x \wedge m) \xrightarrow{\text{shift}} (s+1)) \ominus ((x \wedge m) \xrightarrow{\text{shift}} (s+2)) \ominus \ldots$$

Now using the Logical Function Exclusive-or Theorem we have the next theorem.

Theorem 46 (Degraying)

Let mask m and length s be used to define an extraction of a range of bits: $x[i.j]$ where $x[i.j] = (x \wedge m) \xrightarrow{\text{shift}} s$. Then

$$\Omega(f(degray(x[i.j]))) \leq bc(m)$$

Proof:

$$\begin{aligned} f(degray(x[i.j])) &= f(((x \wedge m) \xrightarrow{\text{shift}} s) \ominus ((x \wedge m) \xrightarrow{\text{shift}} (s+1)) \ominus \\ &\qquad ((x \wedge m) \xrightarrow{\text{shift}} (s+2)) \ominus \ldots) \\ &= \textstyle\sum_{k=0}^{2^L-1} w_k \psi_k(((x \wedge m) \xrightarrow{\text{shift}} s) \ominus ((x \wedge m) \xrightarrow{\text{shift}} (s+1)) \ominus \\ &\qquad ((x \wedge m) \xrightarrow{\text{shift}} (s+2)) \ominus \ldots) \\ &= \textstyle\sum_{k=0}^{2^L-1} w_k \psi_{(m \wedge (k \xrightarrow{\text{shift}} s)) \div (m \wedge (k \xrightarrow{\text{shift}} (s-1))) \div (m \wedge (k \xrightarrow{\text{shift}} (s+2))) \div \ldots}(x) \\ &= \textstyle\sum_{k=0}^{2^L-1} w_k \psi_{m \wedge ((k \xrightarrow{\text{shift}} s) \div (k \xrightarrow{\text{shift}} (s-1)) \div (k \xrightarrow{\text{shift}} (s-2)) \div \ldots)}(x) \end{aligned}$$

Proceeding as with the General Extraction Theorem, it is clear that

$$\Omega(f(degray(x[i.j]))) \leq \min(bc(m), L - s)$$

The argument that $\Omega(f(degray(x[i.j]))) \leq \Omega(f)$ doesn't hold in this case since the series of exclusive-ors may, in fact, increase the bit count (bc). The other limiters still apply.

Since the Degraying Theorem is stated specifically for extraction, we know $bc(m) \leq L - s$. Therefore

$$\Omega(f(degray(x[i.j]))) \leq bc(m)$$

□

The proofs of both the General Extraction Theorem and the Degraying Theorem had the feature that at some point all of the right shift operators were reversed to left shift operators. Having a convenient notation for the resulting left shifted value would be useful. Therefore I define a function called **ungray**:

$$ungray(n) = n \ominus (n \xrightarrow{\text{shift}} 1) \ominus (n \xrightarrow{\text{shift}} 2) \ominus \ldots$$

ungray is just like *degray* except that the shifts are in the opposite direction. This can give binary strings an infinite set of leading 1's. This does not affect the ability to test two of the strings for equality or to perform operations such as logical "and" upon them.

In the next corollary, I simply substitute the *ungray* function into the last summation in the previous proof.

Corollary 46a

If w'_n is the n^{th} Walsh coefficient of the function $f(degray((x \wedge m) \xrightarrow{\text{shift}} s))$ and w_k is the k^{th} Walsh coefficient of the function $f(x)$ then

$$w'_n = \sum_{k\,:\,n=(m \wedge ungray(k \xrightarrow{\text{shift}} s))} w_k$$

Proof:

The proof follows directly from the proof of Degraying Theorem where I noted:

$$f(degray(x[i.j])) = \sum_{k=0}^{2^L-1} w_k \psi_{m \wedge ((k \xrightarrow{\text{shift}} s) \div (k \xrightarrow{\text{shift}} (s+1)) \div (k \xrightarrow{\text{shift}} (s+2)) \div \ldots)}(x)$$

and the definition of the *ungray* function.

□

It is important to notice from the corollary that the nice property of alternating zero and nonzero Walsh sums for odd and even order functions is destroyed by application of degraying. This means if the arguments to an even or odd order function are centered by argument centering, but the arguments are first decoded from Gray code, the even or odd order parity of the function is *not* preserved. However, all is not lost, as we shall see in the next theorem. It turns out that degraying has an interesting affect on odd and even order functions.

Theorem 47 (Even Order Degraying Theorem)

Let mask m and length s be used to define an extraction of a range of bits: $x[i..j]$ where $x[i..j] = (x \wedge m) \xrightarrow{\text{shift}} s$. If $f(x[i..j]) : \mathcal{B}^L \rightarrow \mathcal{R}$ is an even order function then

$$\Omega(f(degray(x[i..j]))) \leq bc(m) - 1$$

Proof:

Again we use the fact that $x[i..j] = m \wedge (x \xrightarrow{\text{shift}} s)$ for appropriately chosen mask m and shift s. If $F = f(x[i..j])$ and $G = f(degray(x[i..j]))$ then by the General Extraction Theorem: if w are the Walsh coefficients of f and w^F are the Walsh coefficients for F:

$$w_n^F = \sum_{k \,:\, n=(m \wedge (k \xrightarrow{\text{shift}} s))} w_k$$

We also know from the corollary to the Degraying Theorem:

$$w_n^G = \sum_{k \,:\, n=(m \wedge ungray(k \xrightarrow{\text{shift}} s))} w_k$$

Because m and s are chosen to represent an extraction of bits, m is a mask consisting of a field of contiguous 1's possibly surrounded on either or both sides by a field of zeros. Notice

that $k \xleftarrow{\text{shift}} s$ has zeros in each position where m has 0's to the right of the field of 1's. Therefore the identity $(m \wedge ungray(k \xleftarrow{\text{shift}} s)) = ungray(m \wedge (k \xleftarrow{\text{shift}} s))$ holds and so:

$$w_n^G = \sum_{k \,:\, n = ungray(m \wedge (k \xleftarrow{\text{shift}} s))} w_k$$

It can be seen that the w^G are simply a remapping by the *ungray* function of the coefficients w^F.

Let's look at this mapping more closely. If $d = ungray(g)$ then the p^{th} bit position in d. denoted $d[p]$ is the parity of all of the bits at that position and to the right.

$$\begin{aligned} n[p] &= ungray(g)[p] \\ &= (g \ominus (g \xleftarrow{\text{shift}} 1) \ominus (g \xleftarrow{\text{shift}} 2) \ominus ...)[p] \\ &= g[p] \ominus (g \xleftarrow{\text{shift}} 1)[p] \ominus (g \xleftarrow{\text{shift}} 2)[p] \ominus ... \\ &= g[p] \ominus g[p-1] \ominus g[p-2] \ominus ... \\ &= \bigoplus_{l=0}^{p} g[l] \end{aligned}$$

If F is an even order function then all $w_n^F = 0$ if $bc(n)$ is odd. If $bc(n)$ is odd then by our observation about the *ungray* function. $ungray(n)$ must have the high bit (left most bit of the masked region set). Hence. all $w_n^G = 0$ if the high bit of n is set and therefore

$$\Omega(f(degray(x[i,j])) \leq bc(m) - 1$$

and the theorem is proved.

□

An analogous proof can be given that shows that for odd order functions all of the nonzero Walsh coefficients of f are mapped to the upper half of the Walsh coefficient space. that is. with the high order bit set. In this case. the Ω of the function is obviously not

reduced by 1 as it is for even order functions.

Corollary 47a

If $f : \mathcal{B}^L \rightarrow \mathcal{R}$ is an even order function then

$$\Omega(f(degray(x)) \leq L - 1$$

Proof:

The proof follows directly from the theorem above.

4.5.4 Generalized Decoding

The next theorem allows us to unite the extraction theorem and polynomial theorem and apply them to the composite function, f_{fitness}, that we are trying to optimize. In the theorem below, the coefficients of the polynomial are indexed by the exponents on the different variables in a standard fashion. The exponent of an absent variable is considered an exponent of 0. An example polynomial of three variables is:

$$P(x, y, z) = a_{123}xy^2z^3 + a_{021}y^2z + a_{004}z^4$$

Theorem 48 (Polynomial Composition)

Let $P_n(x_0, x_1, ...x_{n-1})$ be a polynomial in $x_0, x_1, ...x_{n-1}$ that takes $\mathbb{R}^n \rightarrow \mathbb{R}$ such that

$$P_n(x_0, x_1, ...x_{n-1}) = \sum_{all\ terms} a_{k_0k_1...k_{n-1}} x_0^{k_0} x_1^{k_1} ... x_{n-1}^{k_{n-1}}$$

with $k_i \geq 0$ and let $f_0, f_1, ...f_{n-1}$ be functions such that $f_i : \mathcal{B}^L \rightarrow \mathbb{R}$ then

$$\Omega(P_n(x_0, x_1, ...x_{n-1})) \leq \max_{all\ terms} k_0\Omega(x_0) + k_1\Omega(x_1) + ... + k_{n-1}\Omega(x_{n-1})$$

and

$$\Omega(P_n(f_0(x), f_1(x), ...f_{n-1}(x))) \leq \max_{all\ terms} k_0\Omega(f_0(x)) + k_1\Omega(f_1(x)) + ... + k_{n-1}\Omega(f_{n-1}(x))$$

Proof:

The Ω of any given term is

$$\begin{aligned}
\Omega(P_n(x_0, x_1, ..., x_{n-1})) &= \Omega(\textstyle\sum_{all\ terms} a_{k_0 k_1 ... k_{n-1}} x_0^{k_0} x_1^{k_1} ... x_{n-1}^{k_{n-1}}) \\
&\leq \max_{all\ terms} \Omega(a_{k_0 k_1 ... k_{n-1}} x_0^{k_0} x_1^{k_1} ... x_{n-1}^{k_{n-1}}) \\
&= \max_{all\ terms} \Omega(x_0^{k_0} x_1^{k_1} ... x_{n-1}^{k_{n-1}}) \\
&\leq \max_{all\ terms} \Omega(x_0^{k_0}) + \Omega(x_1^{k_1}) + ... + \Omega(x_{n-1}^{k_{n-1}}) \\
&\leq \max_{all\ terms} k_0 \Omega(x_0) + k_1 \Omega(x_1) + ... + k_{n-1} \Omega(x_{n-1})
\end{aligned}$$

proving the first part of the theorem. The second part follows directly by substitution in the logic above.

4.6 Applying Epistasis Estimation to Fitness Functions

The Polynomial Composition Theorem can be applied to the model of the fitness function we saw in Equation 1.2 :

$$f_{\text{fitness}} = f_{\text{ga}}(f_{\text{decode}}))$$

If f_{ga} can be represented as a polynomial, then the polynomial P_n from the theorem statement is f_{ga}. The decoding function can be represented as a vector of functions $\vec{f} = (f_0, f_1, ..., f_{n-1})$ where $f_i : \mathcal{B}^L \rightarrow \mathcal{R}$ extracts the i^{th} real argument for the model. Therefore for a chromosome x in $\mathcal{B}^L$ and for model functions that are polynomials of n variables:

$$f_{\text{model}}(f_{\text{decode}}(x)) = P_n(f_0(x), f_1(x), ... f_{n-1}(x))$$

Two very important observations can be made. First, given just the mathematical models and extraction functions, we can derive a very good upper bound for the degree

of bit interaction in the f_{fitness} that we are trying to solve. We may be able to apply this understanding to control the level of complexity and thereby improve performance. Second, the theorems we have seen so far suggest that functions which are based on polynomials and extractions of small numbers of bits have a certain inherent simplicity about them. This is because the epistasis of the problems is limited by degree of polynomial and numbers of bits extracted.

In the next section we will give some examples of the predictive power of the theorems. I will estimate the complexity of a common test function that was designed to have fixed levels of bit interactions. Finally, I will use this knowledge to control the complexity of the problem in some simple ways.

4.6.1 Experiment: Predicting Maximum Epistasis

In this section we will give examples of the predictive power of the theorems. We will examine tables of Walsh sums W_0 through W_8 for sample functions. The first column of each table is the order of the Walsh sum. The remaining columns are the Walsh sums of that order for various functions listed at the head of each column. All functions presented in this section are evaluated using a bitstring length of 8, i.e., $L = 8$.

In the first experiment, I demonstrate that the $\Omega(f)$ is bounded by the Polynomial Complexity Theorem to the maximum power of the polynomial. To do this, I computed the Walsh sums of several functions. In Table 4.1, the first column is the order of the Walsh sum. The second column contains the values of the Walsh sums for the simple linear function x, $x \in [0, 2^L - 1]$. The third column contains the Walsh sums for the function x^5. As predicted by the Polynomial Complexity theorem Ω equals the degree of the polynomial.

Our second experiment demonstrates the effect of argument centering. In Table 4.2, I compare the function x^5 over the noncentered domain $[0, 2^L - 1]$ with the centered domain $[-(2^{(L-1)} - .5), (2^{(L-1)} - .5)]$. We see in column 2 that without centering x^5 has nonzero Walsh sums for all orders less than 6, but with centering the even order Walsh sums go to zero as predicted by the Parity Laws and argument centering.

Order	x	x^5
0	127.5	1.811e+11
1	127.5	4.299e+11
2	0	3.457e+11
3	0	1.088e+11
4	0	1.229e+10
5	0	3.731e+08
6	0	0
7	0	0
8	0	0

Table 4.1: Walsh Sums for $f(x) = x$ and x^5

Restricting f_{model} to a polynomial may seem limiting, but actually it is quite powerful since all continuously differentiable functions can be expressed as a Taylor series expansion. For example, the Taylor series expansion about 0 for cos is:

$$\cos(x) = 1 - x^2/2! + x^4/4! - x^6/6! + ...$$

In this case, we see that all the terms of the series are even order polynomials. So cosine has the potential to be an even order function.

In our third experiment, we examine the Walsh sums for $\cos(x)$ over two 8 bit domains. The results are presented in Table 4.3. Again the first column is the order of the Walsh sum. The second column is the Walsh sums for $x \in [0, \pi/2]$. This domain is generated by using a decoding function for the bitstring of $f_{\text{decode}}(z) = z(\pi/2)/(2^8 - 1)$. By centering the argument x so that $x \in [-\pi/2, \pi/2]$, the powers of x in cosine, which are all even, force the function to become even order as we can see in column 3 of the table. To perform the

Order	Without Centering	With Centering
0	1.811e+11	0
1	4.299e+11	1.923e+10
2	3.457e+11	0
3	1.088e+11	1.409e+10
4	1.229e+10	0
5	3.731e+08	3.731e+08
6	0	0
7	0	0
8	0	0

Table 4.2: Walsh Sums for x^5

centering. I used a decoding function of $f_{\text{decode}}(z) = (z - \delta)(\pi/2)/\delta$ where $\delta = 2^7 - .5$. A very important caveat about centering is that one may not always be able to change the domain of arguments to a function, but there is often more flexibility than there may seem at first. A second observation is that if cosine was defined over a 7 bit function rather than 8, then the highest order nonzero Walsh sum for the function would be 6 since $W_7 = 0$. This is a reduction in the Ω for the function by applying centering. We will use this technique in the next section to reduce Ω and other measures of difficulty.

4.6.2 Experiment: Controlling Epistasis with Centering

Griewangk's function is often used as a test function for evolutionary algorithms (Whitley et al., 1995b; Mülenbein and Schlierkamp-Voosen, 1993) A generalized version of the Griewangk function is:

Order	Without Centering	With Centering
0	0.63608	0.63412
1	0.51532	0
2	0.13687	0.64981
3	0.015334	0
4	0.00078529	0.015701
5	1.8782e-005	0
6	2.0600e-007	1.6710e-005
7	9.5998e-010	0
8	1.4848e-012	4.8355e-010

Table 4.3: Walsh Sums for $\cos(x)$

$$G_{d,b}(x) = 1 + \sum_{i=1}^{d} x_i^2/4000 + \prod_{i=1}^{d} \cos(x_i/\sqrt{i})$$

where $x \in \mathcal{B}^{db}$ is a d dimensional vector of arguments with each x_i being b bits wide. For example:

$$x = \overbrace{\underbrace{001101}_{x_1}\underbrace{101100}_{x_2}\underbrace{101101}_{x_3}}^{b \text{ bits } b \text{ bits } b \text{ bits}} \ldots \overbrace{\underbrace{010010}_{x_d}}^{b \text{ bits}}$$

Consider both binary encoding and a Gray encoding. In the case of binary encoding, I will extract the bits of each argument and then shift and scale it. In the case of Gray encoding, a *degray* function is applied between the extraction operation and the shift and scale operation.

Using the theory we have developed, what can we tell about the Walsh coefficients from the formula and encoding alone? Let's begin by computing the maximum number of bits of interaction, Ω, for a *binary encoding*, assuming the arguments are centered:

$$\begin{aligned}
\Omega(G_{d,b}) &= \Omega(1 + \textstyle\sum_{i=1}^{d} x_i^2/4000 + \prod_{i=1}^{d} \cos(x_i/\sqrt{i})\ ) \\
&= \max(\Omega(1),\ \Omega(\textstyle\sum_{i=1}^{d} x_i^2/4000),\ \Omega(\prod_{i=1}^{d} \cos(x_i/\sqrt{i}))\ ) \\
&= \max(0,\ \max_{i=1}^{d}(\Omega(x_i^2/4000),\ \textstyle\sum_{i=1}^{d} \Omega(\cos(x_i/\sqrt{i})))\ ) \\
&= \max(0,\ \max_{i=1}^{d}(\Omega(x_i^2),\ \textstyle\sum_{i=1}^{d} \Omega(\cos(x_i)))\ )
\end{aligned}$$

At this point we can use the Corollary to the General Extraction Theorem and the fact that each x_i is created from an extraction of b bits.

$$\Omega(G_{d,b}) = \max(0,\ \max_{i=1}^{d}(\min(2,b),\ \textstyle\sum_{i=1}^{d} \min(\Omega(\cos(x_i)),b))\ )$$

We note that cosine is an even order function and so only has nonzero Walsh coefficients with indexes having an even number of bits. (Remember cosine is an even order function only when its arguments are centered.) Therefore, if b is even, then $\min(\Omega(\cos(x_i)),b) = b$. If b is odd then $\min(\Omega(\cos(x_i)),b) = b-1$, since the b^{th} Walsh sum is zero and the next nonzero Walsh sum is $b-1$. Let the act of rounding down to the closest number divisible by 2 be denoted by $\lfloor b \rfloor_2$, then $\min(\Omega(\cos(x_i)),b) = \lfloor b \rfloor_2$ and so:

$$\begin{aligned}
\Omega(G_{d,b}) &= \max(0,\ \max_{i=1}^{d} \min(2,b),\ \textstyle\sum_{i=1}^{d} \lfloor b \rfloor_2) \\
&= \max(\min(2,b),\ d\lfloor b \rfloor_2)
\end{aligned}$$

If the function is *not centered* then the analysis simply becomes:

$$\Omega(G_{d.\,b}) = \max(0.\ \max_{i=1}^{d} \Omega(x_i^2).\ \sum_{i=1}^{d} \Omega(\cos(x_i)))$$

$$= \max(0.\ \max_{i=1}^{d} (\min(2.b).\ \sum_{i=1}^{d} b))$$

$$= \max(\min(2.b).\ db)$$

which in most cases is simply db.

Dimension (d)	Width (b)	Ω for Binary Encoding		Ω for Gray Encoding	
		Centered	Offset	Centered	Offset
1	2	2	2	1	2
1	3	2	3	2	3
1	4	4	4	3	4
1	5	4	5	4	5
1	6	6	6	5	6
1	7	6	7	6	7
1	8	8	8	7	8
1	9	8	9	8	9
1	10	10	10	9	10
3	2	6	6	3	6
3	3	6	9	6	9
3	4	12	12	9	12
3	5	12	15	12	15
3	6	18	18	15	18
5	2	10	10	5	10
5	3	10	15	10	15
5	4	20	20	15	20

Table 4.4: Ω for the Generalized Griewangk function, $G_{d.b}$, in both Centered and Offset Forms Using Both Binary and Gray Encodings.

Empirical confirmation of these results is presented in table 4.4. Each row of the table was generated by exhaustively examining all the values of the Griewangk function, $G_{d.b}$, for the given values of d and b, with and without Gray code, and with and without an offset. Recall that Ω is the largest i such that $W_i \neq 0$, thus the odd and even nature of the function impacts Ω. The column labeled "Dimension" refers to the variable d above and represents the number of arguments to the function. "Width" refers to the variable b above and represents the number of bits encoding each argument. Under the heading "Binary Encoding" are the two columns of Ωs. The first column, labeled "Centered", contains the

Ωs for functions whose arguments are centered in the range $[-(2^{b-1} - .5). 2^{b-1} - .5]$. The second column, labeled "Offset", contains the Ωs for functions whose arguments are offset from center by an arbitrary nontrivial constant, $e^{-.5}$, giving a range of $[-(2^{b-1} - .5) + e^{-.5}. 2^{b-1} - .5 + e^{-.5}]$. The affect of parity on the function under a binary encoding shows up as a limit on Ω such that it is rounded down to the nearest even number.

If the same function is used with a Gray encoding then a degraying step must be added by applying the *degray* function **after** extraction and before scaling and offset adjustments. The analysis is similar to the binary centered case except the Corollary to the General Extraction Theorem is replaced by the Even Order Degraying Theorem in the second step below.

$$\begin{aligned}
\Omega(G_{d.\,b}) &= \max(0.\ \max_{i=1}^{d}(\Omega(x_i^2)).\ \sum_{i=1}^{d}\Omega(\cos(x_i))\ ) \\
&= \max(0.\ \max_{i=1}^{d}(\min(2.b)).\ \sum_{i=1}^{d}(b-1)\ ) \\
&= \max(\min(2.b).\ d*(b-1)\ )
\end{aligned}$$

Empirical confirmation of these results is presented in the last two columns of table 4.4. Here it can be seen that the use of cosine in the Griewangk function means that using centering and a Gray encoding is guaranteed to reduce the maximum number of bits of interaction in the function by a number of bits equal to the number of arguments to the function, d. Thus, Gray coding causes a further general reduction in Ω and thus removes the higher level nonlinearities that are present in a representation that is not centered.

In summary: the effects of parity due to centering is shown to reduce Ω. Combining Gray coding with centering can further remove high order nonlinear interactions.

4.6.3 Centering and Problem Difficulty

The classic Griewangk function, as presented in (Whitley et al., 1995b) and (Rana and Whitley, 1997), is a 10 dimensional function with 10 bit fields, one for each argument. The arguments are scaled $[-512.511]$ which is the same as offsetting a centered argument by $-.5$. In (Meysenburg and Foster, 1997; Mühlenbein et al., 1991), the arguments were mapped

range	Optimum Value	Ω	Coverage	Optima Counts: Total Number Min	Optima Counts: Total Number Max	Optima Counts: Global Min	Optima Counts: Global Max	Number Evaluations: Mean	Number Evaluations: Stddev
-15.5. 15.5	0.0558068	16	0.0625	144	144	1	1	2575.3	2307.9
-15.6. 15.4	0.0472153	20	1.0	144	162	1	1	5217.3	3675.5
-15.7. 15.3	0.041276	20	1.0	262	264	1	1	5476.6	3503.6
-15.8. 15.2	0.0312921	20	1.0	400	401	1	1	14754.8	19161.1
-15.9. 15.1	0.0103844	20	1.0	400	480	1	1	13110.3	12428.8
-16.0. 15.0	0	20	1.0	810	810	1	4	10874.2	7943.5

Table 4.5: Results of Six Generalized Griewangk Functions From Centered to Classical

to reals centered on [-600. 600]. From our work in the previous section we would expect argument centering for the Griewangk function would reduce epistasis and possibly increase algorithm performance. To test this hypothesis. I used a genetic algorithm to repeatedly optimize the Griewangk function. I varied the amount of offset form center and measured the performance. Computing the statistics I wanted would require complete enumeration of the function to be sure all regions of the function were explored. For the classic Griewangk function. this would be a space of 2^{100} function values. which is too large to be practical. In order to get around this limitation I reduced the space to 20 bits. This allows me to exhaustively analyze all points in the function domain. The function I choose was a 4 dimensional version of the generalized Griewangk with 5 bits in each dimension. I looked at 6 versions of the function. The first version is an argument centered version with each argument in a range $[-15.5. 15.5]$. The remaining five functions have an increasingly distant offsets from center of $-.1$ to $-.5$. For example. an offset of $-.5$ gives a range from $[-16. 15]$ simulating. in a smaller number of bits. the classical Griewangk which contains zero in the set of possible domain values. All functions in this experiment use a Gray encoding of the arguments.

Table 4.5 shows the results of testing these 6 functions. The first column is the range of the arguments with increasing offset from centered. The second column shows the global optimum of the discretized function. Clearly the optimum must vary given the different sampling of the domain imposed by the different offsets. The third column is the Ω determined by computing the Walsh coefficients based on the 2^{20} values in the function. The

coverage is the ratio of the number of nonzero Walsh coefficients to the total number of Walsh coefficients. The next two columns show the total number of (local and global) minima and maxima. Minima and maxima are defined with respect to a Hamming-1 neighborhood using a Gray code representation of the function. Although the number of local optima using a Hamming-1 neighborhood is indicative of the difficulty facing a GA if it were only using single bit mutation. it is not known how this value relates to the various other GA operators. I supply these columns for comparison with previous work. since number of local optima is often used as an indication of the "ruggedness" of a landscape (Kauffman. 1993: Smith and Smith. 1999). Notice that as the offset increasingly shifts the function from centered. the total number of optima increase. The next column shows the number of global optima. This suggests as this particular function deviates from centered arguments. the number of local optima increase. The number of global optima stays constant at 1.

The last two columns show the time required to locate the global optimum of the function using a genetic algorithm. The algorithm is a steady state (nongenerational) GA similar to GENITOR: parents are selected a pair at a time and a single offspring replaces the worst in a population. Rank based selection is used and no duplicate chromosomes are allowed in the population. The numbers in the table represent the results when solved with a population of 64 individuals. a mutation rate of 0.1 and a linear rank based selection bias probability distribution function with a .1 Y-intercept at the worst individual and a 1.9 at the best. (This selects the best ranked individuals 19 times more frequently than the worst ranked with a linear interpolation for the remaining individuals.)

The number of evaluations measured in the table is the number of times a new child is generated and its fitness evaluated. The last two columns contain the mean and standard deviation of the number of evaluations of the function necessary to find the optimum value over 32 runs for each function. The results indicate that the centered function is easier than the non-centered versions of the function.

The domain of multidimensional real valued functions I am trying to optimize is discrete and often mapped to series of bit fields. This means that each argument to the real valued function can take on one of some power of 2 number of different values. In the case of

Griewangk's function, the centered arguments fail to cover the value zero where the true optima is. The result is that for a Griewangk function, where the domain of each argument is symmetric about zero, centering produces 2^d number of equal valued optima, where d is the dimension of the Griewangk's function.

The reason that the centered function in Table 4.5 does not show the predicted 2^4 optima is that all of the optima are adjacent in Hamming space and hence counted by our software as belonging to one basin of attraction. The existence of the 16 minima in one basin was empirically verified by a separate program.

4.6.4 The Applicability of Centering

The analysis I have developed in this dissertation worked well for Griewangk's function, which is considered to be a nontrivial test function. However, Griewangk was particularly susceptible to our analysis and the use of centering techniques.

From a practical standpoint there are several concerns with applying argument centering to a problem and thereby altering the interval of the domain of the function. The first concern is over rediscretizing the domain of a continuous function in order to center the domain. For instance, a new discretization may not contain the global optimum of the continuous function. Of course, in real world situations the global optimum is usually not known a priori so it is difficult to determine if this case applies. In fact, the new discretization may include a better optimum. Another problem with rediscretizing is that a real world problem specification may require the examination of exactly the solution space given by the precentered discretization. In this case, centering might be achieved by expanding the search space to a symmetric one, but still containing all of the points of uncentered domain. The cost for increasing the size of the search space might be a lengthening of the search in spite of centering. This is an area that needs further research.

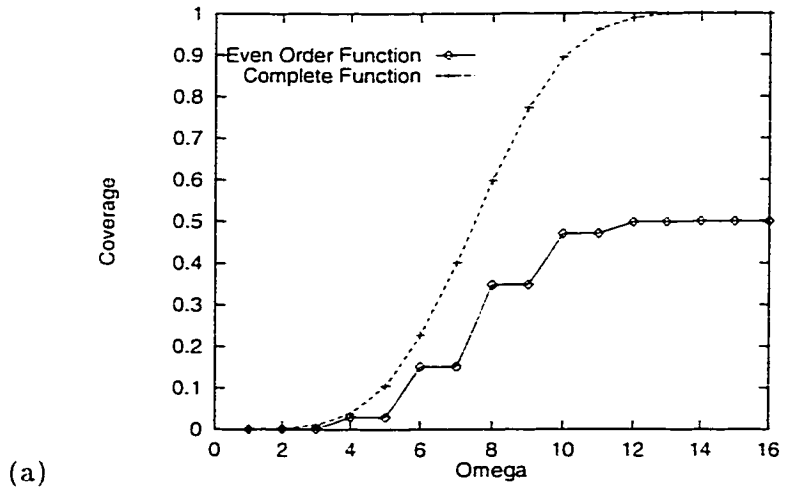
A third concern is that the main benefit of centering arguments is to functions that can be converted to odd or even functions. Straight forward centering can occur only to functions such as sine, cosine or other functions with only even or odd exponents in their polynomial expansions. Whitley (Whitley, 1999) suggests that if the portions of a function

that are major contributors to poor performance are either all even or all odd ordered then this technique may work even though the entire function does not have an even or odd parity.

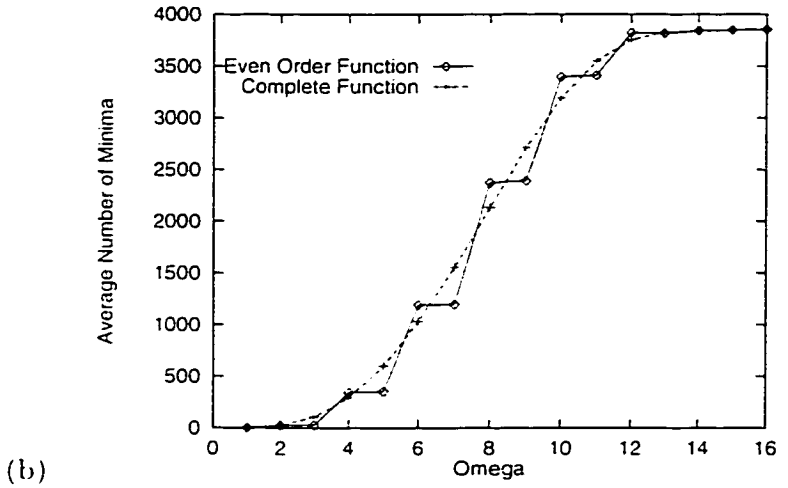
Finally, argument centering introduces symmetry to the function without increasing the number of bits in the representation. By symmetry, I mean the function becomes either even order or odd order and so by the Function Parity Theorem, $f(x) = \pm f(\overline{x})$. In that sense, the search space size is reduced and this offers one possible explanation for the performance increase.

4.6.5 Experiment: The Effects of Selective Walsh Filtering

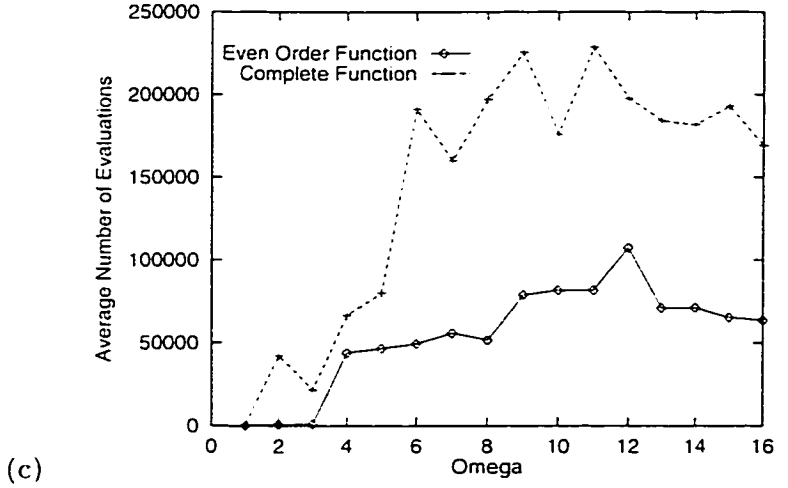
Do the performance results we see for Griewangk hold in general for other functions with either limited Ω or parity such as evenness or oddness? Are these measures a useful indicator of problem difficulty?



(a)



(b)



(c)

Figure 4.1: All Graphs are for 16 Bit Even Order and Complete Functions of Ω from 1 to 16 Bits. The graphs show (a) coverage. (b) average number of minima. (c) average number of evaluations to solution.

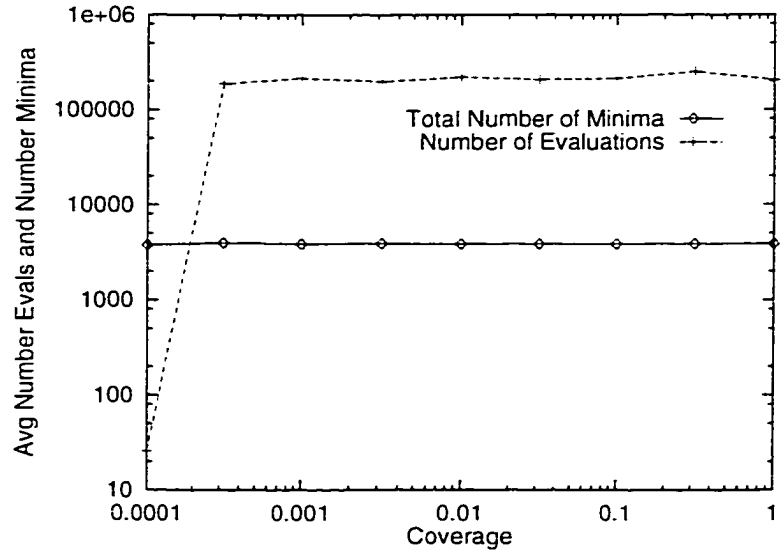


Figure 4.2: A Comparison of Number of Minima and Number of Evaluations for Problems of a Given Level Random Coverage.

To explore these questions. I performed an experiment on two classes of functions: *even order functions* and *complete functions*. In a **complete function**. all the Walsh coefficients whose order is Ω or less (i.e.. has Ω or fewer bits set) are assigned a random value between -1 and $+1$. This practically guarantees all Walsh coefficients of order less than or equal to Ω are nonzero. The remaining coefficients are all zero. The function is created by using the inverse Walsh transform. This generates highly irregular landscapes. but with limited orders of bit interactions. In an **even order function**. all the *even order* Walsh coefficients whose order is Ω or less (i.e. has an even number of bits less than Ω set) are assigned a random value between -1 and $+1$. This effectively zeros out all odd order Walsh sums. The results of this experiment is shown in Figure 4.1.

For each class of function I generated 32 different 16 bit functions. Each problem was then solved in 32 attempts by random populations of 64 chromosomes using the same steady state GA with rank selection as before. This was repeated for maximum Ω's from 1 to 16. The various values of Ω are given on the X-axis of the first three graphs in Figure 4.1.

In Figure 4.1a. both classes of functions behave as one would expect and is a direct consequence of the available number of Walsh coefficients of order less than or equal to Ω.

In the case of the complete function, the curve represents the sum of the number of Walsh coefficients of each order. For an L bit domain this is $\sum_{k=0}^{\Omega} \binom{L}{k}$. You can see the stepwise behavior in the coverage for the even order function since the number of available nonzero Walsh coefficients is the same for $\Omega = 2k$ and $\Omega = 2k+1$. This graph shows how coverage is a direct consequence of Ω.

Figure 4.1b shows the average number of minima in both classes of functions by Ω. In this case, a minimum is defined as either a local minimum or global minimum using a Hamming neighborhood with minimum points that are adjacent in Hamming space being counted as a single minimum. Notice the very strong correlation between coverage and number of minima. This is especially evident for the even order functions where the number of minima displays the same stepwise behavior as the coverage. A very interesting feature is that the number of minima for complete and even order functions are about the same throughout the range of Ω. The removal of the odd order partitions didn't seem to strongly affect the number of minima of the function. This graph shows that the number of minima is strongly related to the Ω.

The difficulty of the functions for the genetic algorithm is measured by number of function evaluations needed to find a solution and is displayed in Figure 4.1c. The number of evaluations seemed to remain fairly high until the Ω drops below half of the number of bits. This is probably partly due to the fact that the potential number of Walsh coefficients, all of approximately the same magnitude, forms a bell curve and so the addition of higher order Walsh coefficients from the tail of the curve has little effect on the final function value. The removal of the odd order Walsh coefficients seems to have curbed the precipitous increase in difficulty as the Ω approached the middle of the range.

These graphs suggest that the Ω may influence coverage, number of minima and problem difficulty. But is the measure of coverage alone a measure of difficulty? To examine this, I performed the same experiment ignoring the maximum number of bits of interaction, Ω, but setting the coverage to fixed values. The results are found in Figure 4.2. Here coverage is on the X-axis. The coverage for each test was reduced by dividing the previous coverage by $\sqrt{10}$ forming a log scale along the X-axis. In each case, the Walsh coefficients that are set

to zero are chosen randomly *without regard to parity.* Surprisingly, the number of minima remain fairly constant, as does the performance measured in number of function evaluations. This suggests that, ignoring cases of extreme depletion of nonzero Walsh coefficients, the difficulty and the number of minima are **not** related to the coverage as long as the nonzero Walsh coefficients are randomly dispersed.

We saw in the third graph of Figure 4.1 that throughout the range of Ω's the performance of the GA on the even order functions is far superior to the performance on complete functions. If the difficulty of the problem for functions from random Walsh coefficients is not related to the level of coverage between the two types of functions, then we must suspect that the difference is in the structure of what has been removed. This suggests that it is far more useful to remove half of the nonzero Walsh coefficients by zeroing the odd (or even) order coefficients than to randomly remove the same number of Walsh coefficients of various orders.

In both this experiment and the previous Griewangk experiment the optimization problem seemed to be made easier by converting the problem to one of odd or even order. This suggests that performance observations made may be generalized. I believe that one of the key factors that caused this reduction in difficulty is related to the fact that whole Walsh sums are zeroed in even order functions while few to none are zeroed the random zeroing.

A final observation is, regardless of the parity or nonparity of the function, the value of Ω seems to only be related to problem difficulty for values of Ω less than half the number of bits in the problem. Beyond this the problems in our experiments seemed to reach a plateau. In fact for our 16 bit cases even an Ω equal to 6 seemed to belong to the plateau set. I believe this is, in part, due to the fact that above half the number of bits the number of available Walsh coefficients decreases exponentially, so that in the case of random Walsh coefficients the combined affects of the higher order Walsh coefficients are greatly reduced.

4.7 Summary

In the beginning of this chapter I introduced several Walsh based measures of epistasis. I developed numerous theorems on the Walsh analysis of polynomials. Polynomials is not as

limited a problem set as it might first appear. since all continuously differentiable functions can be expressed as a Taylor series expansion and many other problems can be approximated by a finite series of polynomial terms. We saw that polynomials and the various forms of encoding and extraction. constrained that maximum the maximum number of bits of epistasis. Specifically. epstasis for a polynomial had to be less than the degree of the polynomial and less than the number of bits used to extract the argument for the function.

Many researchers suggest that difficulty may be related to epistasis. If this is so. a reduction in epistasis could be used to reduce the difficulty of problems. In the last half of this chapter. I examined some patterns of epistasis. In particular even and odd ordered functions. I discovered that. for samples from several classes of problems. the arrangement of the subsets of interacting bits in the chromosome is important in influencing the ease with which an optimization problem can be solved. In particular. I show that for the Griewangk function. a classic GA test problem. argument centering reduces epistasis and problem difficulty is reduced.

Chapter 5

The Walsh Structure of Logical Expressions

In the last chapter. we saw how polynomials of limited degree produced functions of limited epistasis. In this chapter. we look at a second class of functions of limited epistasis. the class of logical expressions. This may seem like an exercise in pure mathematics. However. it has direct application to a very famous set of problems in computer science know as the k-satisfiability problems. I also show how problems that might not normally be thought of as logical expressions can also be analyzed with the theorems in this chapter. This shows these theorems have greater reach than it might first appear.

Specifically. I develop techniques for determining the Walsh coefficients and hence epistasis of logical expressions based on AND's. OR's. and EXCLUSIVE-OR's. I show that sums of disjunctive or conjunctive clauses and of limited clause size have limited epistasis. I discuss and give examples of using logical expressions as optimization problems. The theorems proven in the first part of the chapter are used to analyze a form of k-SAT problem called MAXSAT in last part of the chapter.

The mathematics developed in this chapter will then be used in the next chapter to further analyze MAXSAT and compare it to another famous problem called the NK-landscape. This will lead to revelations on problems of limited epistasis in general.

5.1 Normal Forms and Satisfiability

All logical expressions can be placed in both disjunctive normal form and conjunctive normal form (Hohn. 1969: Mendelson. 1970: Cormen et al.. 1990). An expression is in **disjunctive normal form (DNF)** if it is a disjunction of **clauses** that are each a conjunction of variables or the negation of variables. For example:

$$(\overline{a} \wedge b) \vee (c) \vee (a \wedge \overline{a})$$

An expression is in **conjunctive normal form (CNF)** if it is a conjunction of clauses that are each a disjunction of variables or the negation of variables. For example:

$$(a \vee \overline{b} \vee c) \wedge (b) \wedge (a \vee \overline{a})$$

A CNF can be trivially converted into a DNF (Hohn. 1969) and vice versa since AND distributes over OR and OR over AND. In the case of translating a CNF to a DNF. the clauses in the resulting DNF are the set of all possible combinations of elements taken one from each clause in the CNF. If there are c clauses in the CNF then there are at most c variables in any clause in the equivalent DNF. Also notice that the number of clauses in the equivalent unsimplified DNF is equal to the product of the number of variables in each clause in the CNF. The above CNF translates into the following DNF:

$$(a \wedge b \wedge a) \vee (a \wedge b \wedge \overline{a}) \vee (\overline{b} \wedge b \wedge a) \vee (\overline{b} \wedge b \wedge \overline{a}) \vee (c \wedge b \wedge a) \vee (c \wedge b \wedge \overline{a})$$

A subclass of CNF form called **k-SAT expression** is defined as a CNF with three constraining parameters: there is a universe of V different variables possible for the formula. of these. an expression is created consisting of c clauses. each clause having k variables. For this dissertation. I will add the constraint that each clause must contain exactly k *different* variables. An example of a 3-SAT expression from 26 variable space denoted by the Latin alphabet and using 5 clauses would be:

$$(\bar{l} \vee i \vee \bar{t}) \wedge (\bar{t} \vee \bar{l} \vee e) \wedge (m \vee \bar{i} \vee s) \wedge (s \vee a \vee \bar{b}) \wedge (b \vee \bar{e} \vee \bar{y}) \qquad (5.1)$$

Notice that all 26 variables were not used.

A **k-satisfiability problem** is the problem of assigning logical values (TRUE or FALSE) to the variables for a specified k-SAT expression that **satisfies the expression**. that is. makes the expression true. A logical expression is ill formed for a stochastic optimizer since it only can have one of two values: TRUE or FALSE. The stochastic optimizer needs at least to be able to tell when one potential solution is closer to the optimum than another: otherwise. the terrain is essentially flat and featureless. This plateau structure will come back to plague us later in this chapter.

A popular "fix" is a fitness function that returns a number from 0 to c. representing the number of clauses satisfied. Of course. this fitness function assumes that it is *better* to have more clauses satisfied than fewer in intermediate values of the search. This formulation for the problem is called a **MAXSAT** problem (Papadimitriou. 1994). A MAXSAT problem with k variables in each clause is a **MAXkSAT** problem. This is an important class of problems in computation theory. As an example of a MAXSAT problem. Equation 5.1 would become the MAX3SAT problem:

$$(\bar{l} \vee i \vee \bar{t}) + (\bar{t} \vee \bar{l} \vee e) + (m \vee \bar{i} \vee s) + (s \vee a \vee \bar{b}) + (b \vee \bar{e} \vee \bar{y})$$

which assumes that TRUE evaluates as 1 and FALSE evaluates as 0. The optimum value. in this case. is 5. An alternative measure might grade the difficulty of each clause by the degree to which the variables in the clause are constrained. The resulting fitness function would be the sum of the grades of the satisfied clauses. The remaining sections develop the Walsh analysis for logical expressions and various means of combining them. In the following chapter. we will analyze the general MAXSAT problem using what we learn here.

5.2 The Walsh Coefficients for Logical Expressions

In this section we will show how to compute the Walsh coefficients for logical expressions. We begin with a very useful general theorem that is more of a technique than a theorem but I want to draw attention to how and why it works.

Theorem 49 (Logical Function Summation)

Given $f : \mathcal{B}^L \rightarrow \mathcal{B}$ then

$$\sum_{k=0}^{2^L-1} f(k)\psi_j(k) = \sum_{k\ :\ f(k)=1} \psi_j(k)$$

Proof:

Since f is returns either a 0 or a 1. clearly:

$$\sum_{k=0}^{2^L-1} f(k)\psi_j(k) = \sum_{k\ :\ f(k)=1} f(k)\psi_j(k) + \sum_{k\ :\ f(k)=0} f(k)\psi_j(k) = \sum_{k\ :\ f(k)=1} 1\psi_j(k)$$

The Logical Function Summation Theorem can be used to find the Walsh coefficients for the two basic cases of single bit functions. Note the use of hyperplane numbering to simplify the proof.

Theorem 50 (Single Bit Functions)

If $f(x) = x[i]$ and $g(x) = \overline{x[i]}$ are L bit functions. where i indexes a single bit in the bitstring x. then:

$$w_j^f = \begin{cases} .5 & \text{if } j = 0 \\ -.5 & \text{if } j = 2^i \\ 0 & \text{otherwise} \end{cases} \qquad w_j^g = \begin{cases} .5 & \text{if } j = 0 \\ .5 & \text{if } j = 2^i \\ 0 & \text{otherwise} \end{cases}$$

Proof:

Given that $f(x) = x[i]$. the Walsh coefficients for f can be computed as follows:

$$\begin{aligned} w_j^f &= \tfrac{1}{2^L} \textstyle\sum_{k=0}^{2^L-1} f(k)\psi_j(k) \\ &= \tfrac{1}{2^L} \textstyle\sum_{k=0}^{2^L-1} k[i]\psi_j(k) \\ &= \tfrac{1}{2^L} \textstyle\sum_{k:k[i]=1} \psi_j(k) \end{aligned}$$

but this constraint selects a single hyperplane...

$$= \tfrac{1}{2^L} \textstyle\sum_{k \in h_{2^{l-1}}} \psi_j(k)$$

Therefore by the Balanced Sum Theorem for Numbered Hyperplanes:

$$w_j^f = \begin{cases} 0 & \text{if } j \not\subseteq 2^l \\ \frac{1}{2^L}\psi_j(2^l)2^{L-1} & \text{if } j \subseteq 2^l \end{cases}$$

giving:

$$w_j^f = \begin{cases} 0 & \text{if } j \not\subseteq 2^l \\ \frac{1}{2}\psi_j(2^l) & \text{if } j = 0 \text{ or } 2^l \end{cases}$$

The values of w^g follow similarly.

A **conjunctive clause** is a logical expression containing only AND and NOT operators. A **disjunctive clause** is logical expression containing only OR and NOT operators. In the following theorems. the operands to the ANDs or ORs and the variables which are negated in these clauses can be denoted with two masks. Consider one of these clauses over a domain of L variables whose truth values are specified by a vector in $\mathcal{B}^L$. A **variable selection mask** m. $m \in \mathcal{B}^L$. selects which bit positions. and hence which variables from the domain. are to be ORed or ANDed. This bit mask approach forces each variable to occur at most once in a clause. A **negation mask** n. $n \in \mathcal{B}^{bc(m)}$. indicates which of the selected bits are negated before use. Note that the length of the negation mask is determined by the number

of variables selected. To see how this works, let $x \in \mathcal{B}^{10}$, then, remembering that the right most bit is the 0^{th} bit, $x[7] \vee x[3] \vee \overline{x[1]}$ can be defined as a disjunctive clause with a variable selection mask of 0010001010 and a **negation mask** of 001. For brevity, a conjunctive clause with selection mask m and negation mask n is denoted $C(m, n)$. A disjunctive clause is similarly denoted $D(m, n)$. Our example clause is: $D(0010001010, 001)$.

Theorem 51 (Conjunction of Bits Without Negation)

If f is an L bit function such that $f(x) = x[a_1] \wedge x[a_2] \wedge x[a_3] \wedge ... \wedge x[a_k]$ where each a_i indexes a single bit in x and, all of the a_i are unique and represented in a variable selection mask m, that is, if $f = C(m, 0)$, then:

$$w_j = \begin{cases} 0 & \text{if } j \not\subseteq m \\ \frac{1}{2^{bc(m)}} \psi_j(m) & \text{if } j \subseteq m \end{cases}$$

Proof:

We know that $bc(m) = k$ and given that $f(x) = x[a_1] \wedge x[a_2] \wedge x[a_3] \wedge ... \wedge x[a_k]$, the Walsh coefficients for f can be computed as follows:

$$\begin{aligned} w_j &= \frac{1}{2^L} \sum_{i=0}^{2^L-1} f(i)\psi_j(i) \\ &= \frac{1}{2^L} \sum_{i:(i[a_1]=1)\wedge(i[a_2]=1)\wedge...\wedge(i[a_k]=1)} 1\psi_j(i) \\ &= \frac{1}{2^L} \sum_{i \in h_{2^{a_1}+2^{a_2}+...+2^{a_k}, 2^k-1}} 1\psi_j(i) \\ &= \frac{1}{2^L} \sum_{i \in h_{m, 2^{bc(m)}-1}} \psi_j(i) \end{aligned}$$

Therefore, by the Balanced Sum Theorem for Numbered Hyperplanes:

$$w_j = \begin{cases} 0 & \text{if } j \not\subseteq m \\ \frac{1}{2^L} \psi_j(m) 2^{L-bc(m)} & \text{if } j \subseteq m \end{cases}$$

For pedagogical reasons, in the previous theorem, I did not consider functions in which any of the bits are negated. The addition of negation is now a simple extension by adding the negation mask n to the $C(m,0)$ of the previous theorem.

Theorem 52 (Conjunction of Bits With Negation)

If f is an L bit function $C(m,n)$, then:

$$w_j = \begin{cases} 0 & \text{if } j \not\subseteq m \\ \frac{1}{2^{bc(m)}} \psi_j(unpack(\overline{n}, m)) & \text{if } j \subseteq m \end{cases}$$

Proof:

The proof is a direct extension of the previous theorem. I will use the indices b_i to indicate which bit positions are negated. Let $f(x) = x[a_1] \wedge x[a_2] \wedge ... \wedge x[a_k] \wedge \overline{x[b_1]} \wedge \overline{x[b_2]} \wedge ... \wedge \overline{x[b_n]}$ and all of the a_i and b_i are unique.

$$\begin{aligned} w_j &= \tfrac{1}{2^L} \textstyle\sum_{i=0}^{2^L-1} f(i)\psi_j(i) \\ &= \tfrac{1}{2^L} \textstyle\sum_{i:(i[a_1]=1)\wedge(i[a_2]=1)\wedge...\wedge(i[a_k]=1)\wedge(i[b_1]=0)\wedge(i[b_2]=0)\wedge...\wedge(i[b_n]=0)} 1\psi_j(i) \\ &= \tfrac{1}{2^L} \textstyle\sum_{i \in h_{2^{a_1}+2^{a_2}+...+2^{a_k}+2^{b_1}+2^{b_2}+...+2^{b_n}, \overline{n}}} 1\psi_j(i) \\ &= \tfrac{1}{2^L} \textstyle\sum_{i \in h_{m,\overline{n}}} \psi_j(i) \end{aligned}$$

In the hyperplane numbering above, $\overline{n}$ is used to turn on the bits that correspond to the a_i's and zero the bits corresponding to the b_i's. By the Balanced Sum Theorem for Numbered Hyperplanes:

$$w_j = \begin{cases} 0 & \text{if } j \not\subseteq m \\ \frac{1}{2^L} \psi_j(unpack(\overline{n}, m)) 2^{L-bc(m)} & \text{if } j \subseteq m \end{cases}$$

and the theorem is proved.

Now we have the theorems we need to neatly prove the Disjunction of Bits Theorem.

Theorem 53 (Disjunction of Bits)

If f is an L bit disjunctive clause $D(m.n)$, then:

$$w_j = \begin{cases} 1 - \frac{1}{2^{bc(m)}} & \text{if } j \subseteq m \text{ and } j = 0 \\ -\frac{1}{2^{bc(m)}} \psi_j(unpack(n.m)) & \text{if } j \subseteq m \text{ and } j \neq 0 \\ 0 & \text{if } j \not\subseteq m \end{cases}$$

Proof:

Any function which can be written as a strict disjunction of terms can be rewritten using De Morgan's Laws as the negation of a conjunction of the negation of the terms. For example: $A \vee B$ can be rewritten as $\overline{\overline{A} \wedge \overline{B}}$. Then by De Morgan's Laws, $D(m.n)$ can be rewritten as the negation of the conjunction with selection mask m and negation mask $\overline{n}$ and denoted $\overline{C(m.\overline{n})}$. Negation of a function can be performed by subtracting the numeric truth value of the function from 1. Therefore:

$$f = D(m.n) = \overline{C(m.\overline{n})} = 1 - C(m.\overline{n})$$

We take the Walsh transform, $\mathcal{W}$, of both sides:

$$\mathcal{W}(f) = \mathcal{W}(1) - \mathcal{W}(C(m.\overline{n}))$$

The Walsh coefficients $\mathcal{W}(1)$ are zero everywhere but w_0 which is 1. The Walsh coefficients $\mathcal{W}(C(m.\overline{n}))$ can be determined by the Conjunction of Bits with Negation Theorem. The expression that follows is the sum of the Walsh coefficients for the two transforms.

$$w_j = \begin{cases} 1 - \frac{1}{2^{bc(m)}}\psi_0(unpack(\overline{\overline{n}}.m)) & \text{if } j \subseteq m \text{ and } j = 0 \\ 0 - \frac{1}{2^{bc(m)}}\psi_j(unpack(\overline{\overline{n}}.m)) & \text{if } j \subseteq m \text{ and } j \neq 0 \\ 0 + 0 & \text{if } j \not\subseteq m \end{cases}$$

Which simplifies to:

$$w_j = \begin{cases} 1 - \frac{1}{2^{bc(m)}} & \text{if } j \subseteq m \text{ and } j = 0 \\ -\frac{1}{2^{bc(m)}}\psi_j(unpack(n.m)) & \text{if } j \subseteq m \text{ and } j \neq 0 \\ 0 & \text{if } j \not\subseteq m \end{cases}$$

We have shown how to compute the Walsh coefficients for conjunctive clauses and disjunctive clauses. The Walsh coefficients for CNFs and DNFs where the logical value of the clauses are added together is simply the addition of the Walsh coefficients for the clauses. In the particular case of a MAXSAT problem, which is a sum of disjunctive clauses, we get this corollary.

Corollary 53a

Let f be a MAXkSAT expression of c clauses with variable selection masks $m_1, m_2, \ldots, m_c$ and negation masks $n_1, n_2, \ldots, n_c$ then

$$w_j^f = \begin{cases} c(1 - \frac{1}{2^k}) & \text{if } j = 0 \\ \frac{1}{2^k}\sum_{i\,:\,j\subseteq m_i}\psi_j(unpack(n_i.m_i)) & \text{if } j > 0 \end{cases}$$

Proof:

The corollary is arrived at by a straightforward sum of the coefficients for each clause being sure to only select coefficients that might be nonzero. This is done by selecting to add in values for clause i only if the subscript j is contained in mask m_i.

Our theorems show that the *magnitude* of bit interactions for both disjunctive and conjunctive clauses are all either 0 or $\frac{1}{2^{bc(m)}}$ except at w_0. This means that if the number of variables is the same in all the clauses. then in the summation of the Walsh coefficients for the clauses of the CNF and DNF there is a lot of opportunity for coefficients to cancel. Soraya Rana made the following conjecture about Walsh coefficient cancellation in the MAXkSAT problem. I provide a proof for it here.

Theorem 54 (Soraya's Conjecture on Walsh Cancellation)

If f is a MAXkSAT Landscape and there are an equal number of negated and nonnegated instances of the n^{th} variable in the MAXkSAT expression associated with the landscape then $w_j = 0$ where $j = 2^n$.

Proof:

We know:

$$w_j^f = \frac{1}{2^k} \sum_{i \,:\, j \preceq m_i} \psi_j(unpack(n_i, m_i))$$

We also know by the definition of the Walsh coefficient that the sign of the ψ_j in the above equation is dependent solely on the parity of $j \wedge unpack(n_i, m_i)$. Since j is a power of two. the value of the Walsh coefficients are equal to -1. if the j^{th} variable is negated: and $+1$. if the variable is not negated. Therefore. if negated and nonnegated variables come in equal numbers in the clauses. then they cancel out. and the Walsh coefficient is zero.

An EXCLUSIVE-OR clause can be defined by two masks similarly to disjunctive and conjunctive clauses. I will denote the EXCLUSIVE-OR clause $X(m, n)$. Since parity is an important notion for Walsh functions we can anticipate an EXCLUSIVE-OR theorem may prove useful. Note that the **parity function.** returns the parity of the bits in its argument. The parity function can be represented as $X(\vec{1}, 0)$.

Theorem 55 (Walsh Coefficients for Parity Function)

If f is the L bit parity function *parity* then

$$w_j^f = \begin{cases} \frac{1}{2} & \text{if } j = 0 \\ -\frac{1}{2} & \text{if } j = 2^L - 1 \\ 0 & \text{otherwise} \end{cases}$$

Proof:

The proof follows directly from the Logical Function Summation Theorem and the Balanced Sum for Parity Theorem.

$$\begin{aligned} w_j &= \tfrac{1}{2^L} \textstyle\sum_{i=0}^{2^L-1} f(i)\psi_j(i) \\ &= \tfrac{1}{2^L} \textstyle\sum_{i=0}^{2^L-1} parity(i)\psi_j(i) \\ &= \tfrac{1}{2^L} \textstyle\sum_{i:parity(i)=1} \psi_j(i) \\ &= \tfrac{1}{2^L} \begin{cases} 2^{L-1} & \text{if } j = 0 \\ -2^{L-1} & \text{if } j = 2^L - 1 \\ 0 & \text{otherwise} \end{cases} \\ &= \begin{cases} \frac{1}{2} & \text{if } j = 0 \\ -\frac{1}{2} & \text{if } j = 2^L - 1 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

□

Now we can prove the general EXCLUSIVE-OR of Bits Theorem by using embedding and reflection of the *parity* function.

Theorem 56 (EXCLUSIVE-OR of Bits)

If f is an L bit EXCLUSIVE-OR clause $X(m, n)$ then

$$w_j = \begin{cases} \frac{1}{2} & \text{if } j = 0 \\ -\frac{1}{2}\psi_j(unpack(n.m)) & \text{if } j = m \\ 0 & \text{otherwise} \end{cases}$$

Proof:

Let $f(x)$ be the function $X(m.n)(x)$. that is the application of the function $X(m.n)$ to the bitstring x.

$$\begin{aligned} X(m.n)(x) &= X(\vec{1}.n)(pack(x.m)) \\ &= X(\vec{1}.0)(pack(x.m) \ominus n) \end{aligned}$$

Therefore. $X(m.n)(x)$ is simply the embedding of the function $X(\vec{1}.n)$ using mask m where $X(\vec{1}.n) : \mathcal{B}^{bc(m)} \rightarrow \mathcal{B}$. This can be further reduced to an embedding of the function: $X(\vec{1}.0)$ reflected about n. But. $X(\vec{1}.0)$ is just the parity function. Let $g(x)$ be $X(\vec{1}.0)$. So working backwards from the Walsh Coefficients for Parity Function Theorem:

$$w_k^g = \begin{cases} \frac{1}{2} & \text{if } k = 0 \\ -\frac{1}{2} & \text{if } k = 2^{bc(m)} - 1 \\ 0 & \text{otherwise} \end{cases}$$

Then using the Embedding with Reflection Theorem:

$$w_j^f = \begin{cases} \frac{1}{2}\psi_0(unpack(n.m) & \text{if } j = 0 \\ -\frac{1}{2}\psi_j(unpack(n.m) & \text{if } j = m \\ 0 & \text{otherwise} \end{cases}$$

which immediately simplifies to the required equation.

The way I stated the theorem was for easy comparison with the disjunctive and conjunctive bit theorems. The expression could be further simplified by noting the Walsh function is just performing a parity check of the negation mask.

The important thing to see here is that all but two of the Walsh coefficients are zero. That means that the EXCLUSIVE-OR clause is the fundamental function that sets the Walsh coefficient whose index equals exactly the variable mask. Hence, given the set of Walsh coefficients for a function, one can reconstruct the function as a linear weighted sum of EXCLUSIVE-OR clauses with w_0 adjusted by adding a constant. But, this is not surprising, since if we return to the definition of the Walsh function:

$$w_j(x) = Y(\bigoplus_{i=0}^{L-1}(x[i] \wedge j[i]))$$

we see that the Walsh function is an EXCLUSIVE-OR of the bits selected from x by mask j. This is:

$$w_j(x) = X(j,0)(x)$$

5.3 Combining Logical Functions

In this section, we discuss briefly the combining of logical functions.

Theorem 57 (Disjunction of Functions)

If $f(x) = f_1(x) \vee f_2(x) \vee \ldots \vee f_m(x)$ where the f_i are logical functions, then

$$w_n^f = \sum_{1 \le i \le m} w_n^{f_i} - \sum_{1 \le i < j \le m} w_n^{f_i \wedge f_j} + \sum_{1 \le i < j < k \le m} w_n^{f_i \wedge f_j \wedge f_k} - \cdots + (-1)^m w_n^{f_1 \wedge f_2 \wedge \ldots \wedge f_m}$$

Proof:

We begin with the Logical Function Summation Theorem applied to our disjunction of functions.

$$
\begin{aligned}
w_j &= \tfrac{1}{2^L} \textstyle\sum_{i=0}^{2^L-1} f(i)\psi_j(i) \\
&= \tfrac{1}{2^L} \textstyle\sum_{i=0}^{2^L-1} (f_1(i) \vee f_2(i) \vee \ldots \vee f_m(i))\ \psi_j(i) \\
&= \tfrac{1}{2^L} \textstyle\sum_{i:f_1(i)\vee f_2(i)\vee\ldots\vee f_m(i)=1} \psi_j(i)
\end{aligned}
$$

But this is really a problem of summing the $\psi_j(i)$ for each i where the OR of the functions f_i is true. This is equivalent to the classic counting problem solved by the Inclusion/Exclusion Principle of combinatorics (Niven, 1965), giving us:

$$
\begin{aligned}
w_j^f = \ & \tfrac{1}{2^L}\Big(\textstyle\sum_{1\le i\le m}\Big(\sum_{z:f_i(z)=1} \psi_j(z)\Big) - \\
& \textstyle\sum_{1\le i<j\le m}\big(\sum_{z:f_i(z)\wedge f_j(z)=1} \psi_j(z)\big) + \\
& \textstyle\sum_{1\le i<j<k\le m}\big(\sum_{z:f_i(z)\wedge f_j(z)\wedge f_k(z)=1} \psi_j(z)\big) - \\
& \ldots \\
& (-1)^m \textstyle\sum_{z:f_i(z)\wedge f_j(z)\wedge\ldots\wedge f_m(z)=1} \psi_j(z)\big)
\end{aligned}
$$

or:

$$
\begin{aligned}
w_j^f = \ & \textstyle\sum_{1\le i\le m}\Big(\frac{1}{2^L}\sum_{z:f_i(z)=1} \psi_j(z)\Big) - \\
& \textstyle\sum_{1\le i<j\le m}\big(\frac{1}{2^L}\sum_{z:f_i(z)\wedge f_j(z)=1} \psi_j(z)\big) + \\
& \textstyle\sum_{1\le i<j<k\le m}\big(\frac{1}{2^L}\sum_{z:f_i(z)\wedge f_j(z)\wedge f_k(z)=1} \psi_j(z)\big) - \\
& \ldots \\
& (-1)^m\big(\frac{1}{2^L}\textstyle\sum_{z:f_i(z)\wedge f_j(z)\wedge\ldots\wedge f_m(z)=1} \psi_j(z)\big)
\end{aligned}
$$

and the theorem follows.

The following corollaries are based directly on the set based reasoning in the previous theorem. Repeated application of the first corollary can be used to prove the theorem. Using that same reasoning, the second corollary can be used to prove an EXCLUSIVE-OR of Functions theorem, but that is left as an exercise to the reader.

Corollary 57a

If $f(x) = a(x) \vee b(x)$, where a and b are logical functions, then

$$w_j^f = w_j^a + w_j^b - w_j^{a \wedge b}$$

where $w^{a \wedge b}$ is the Walsh coefficient for the conjunction of the two functions a and b.

Corollary 57b

If $f(x) = a(x) \ominus b(x)$, where a and b are logical functions, then

$$w_j^f = w_j^a + w_j^b - 2w_j^{a \wedge b}$$

Even though the next corollary is a restatement of the more general theorem that is true for any function a and b, we include it here to draw comparisons with the previous two corollaries.

Corollary 57c

If $f(x) = a(x) + b(x)$ where a and b are logical functions then

$$w_j^f = w_j^a + w_j^b$$

This last corollary says that if the functions are special cases in which there is no epistatic interaction between them then all the higher order sums in the Disjunction of Functions go

to zero.

Corollary 57d

If $f(x) = f_1(x) \vee f_2(x) \vee \ldots \vee f_k(x)$ where the f_i are logical functions and $\forall\ i.j \in \{1.2.\ldots.k\}.i \neq j$ that $f_i(x) \wedge f_j(x) = 0\ \ \forall\ x \in \mathcal{B}^L$ then

$$w_n^f = \sum_{i=1}^{k} w_n^{f_i}$$

Proof:

This follows immediately from the Disjunction of Functions Theorem since all interaction terms are now zero.

5.4 Final Observations about Logical Expressions

With the above theorems. it is now easy to compute the Walsh coefficients and the maximum number of bits of interaction (Ω) for the various important classes of logical expressions. In the case were the function is the arithmetic sum of the logical value 0. 1 from each of the clauses. the maximum number of bits of interaction is the maximum number of variables in any one clause. In particular. in the case of a MAXkSAT the value is k. In the case of pure CNFs or DNFs the maximum number of bits of interaction is the number of distinct variables in all of the clauses. Given any logical expression where the values of the variables are extracted from a bit string one can compute the Walsh coefficients as follows. First convert the logical expression to DNF. Then use the Disjunction of Functions Theorem to combine the clauses into conjunctive clauses. For each of the conjunctive clauses use the Conjunction of Bits with Negation Theorem to derive the Walsh coefficients. The resulting Walsh coefficients are summed as prescribed by the Disjunction of Functions Theorem. The shows that the Walsh coefficients of an arbitrary logical expression are the sum of the Walsh coefficients of conjunctive clauses of the expression in DNF minus interaction terms between pairs of clauses plus interaction terms between triples of clauses etc. Note that this quickly becomes a computational burden if there are a large number of clauses. This also means

that maximum number of bits of epistasis for a given arbitrary logical expression is equal to the number of distinct variables in the expression. However, note that the magnitude of the Walsh coefficients for interactions between clauses decreases as $1/2^n$, where n is the number of distinct variables in the interacting clauses.

One of the most interesting observations about these theorems is the fact that the only difference in Walsh space between $C(m,n)$ and $D(m,n)$ is the sign of the Walsh coefficients and the magnitude of w_0. This is, as I have shown, is a consequence of De Morgan's Laws.

5.5 An Application of Logical Expression Analysis

The logical expression theorems in this chapter are much more versatile than they may appear at first. Some functions that we want to analyze may not appear to logical functions but can be recaste as functions involving logical expressions. The bitcount function, bc, is an example.

Theorem 58 (Walsh Coefficients for Bitcount)

If f is the L bit bitcount function bc then:

$$w_k^f = \begin{cases} \frac{L}{2} & \text{if } k = 0 \\ -\frac{1}{2} & \text{if } k = 2^j \text{ for some } j \\ 0 & \text{otherwise} \end{cases}$$

Proof:

f can be recaste as a MAXSAT problem by simply adding up the value of each bit position:

$$f(x) = \sum_{j=0}^{L-1} C(2^j, 0)$$

If $g_j = C(2^j, 0)$, then we know by The Conjunction of Bits with Negation Theorem:

$$w_k^{g_j} = \begin{cases} 0 & \text{if } k \not\subseteq 2^j \\ \frac{1}{2^{bc(2^j)}} \psi_k(unpack(\overline{0}.2^j)) & \text{if } k \subseteq 2^j \end{cases}$$

which reduces to

$$w_k^{g_j} = \begin{cases} 0 & \text{if } k \not\subseteq 2^j \\ \frac{1}{2}\psi_k(2^j) & \text{if } k \subseteq 2^j \end{cases}$$

Note that this sets both w_0 and w_{2^j} to nonzero values. Therefore, since

$$w_k^f = \sum_{j=0}^{L-1} w_k^{g_j}$$

combining the L Walsh coefficient expressions and recognizing that $\psi_k(2^j) = -1$ when $k = 2^j$:

$$w_k^f = \begin{cases} \frac{L}{2} & \text{if } k = 0 \\ -\frac{1}{2} & \text{if } k = 2^j \text{ for some } j \\ 0 & \text{otherwise} \end{cases}$$

□

We could have deduced that this would be the answer based on the bitwise linearity of the function and that w_0^f is the average of the function values but, this demonstrates that the recasting of a function as a logical expression can be a very powerful tool. This is not the last time we will use this trick.

5.6 Summary

This chapter covered the detailed analysis of logical expressions including an introduction to the famous MAXSAT problem. We saw that bits that were interconnected by AND's or

OR's tended to have epistatic interactions amongst all the bits. However, when clauses were connected by plus then the Walsh coefficients are summed so the maximum epistasis (Ω) is limited to the maximum number of distinct variables in the clauses. Hence, the Ω of a MAXkSAT problem is k. A procedure was outlined for computing the Walsh coefficients of any logical expression via the theorems from this chapter. We also saw that bits that were interconnected by EXCLUSIVE-ORś could be used to select one specific interaction. Finally, we saw that sometimes recasting a problem as a logical expression gives us a powerful tool for Walsh analysis.

Chapter 6

Landscapes of Limited Epistasis

Chapter 3 introduced the concept of embedding a function in a higher dimensional space. The result was a function whose epistasis was limited in comparison to the full size of the domain. If a function f has epistasis such that $\Omega(f) < dim(f)$. then I called the function a **function of limited epistasis**. In chapter 4 we saw that polynomials of limited degree or functions whose arguments where extracted with a limited number of bits would tend to have limited epistasis. In the chapter on logical functions we saw that bounded clause size produced functions of limited epistasis for functions that were sums of logical clauses. I have proven many properties about the epistasis of two major classes of problems: polynomials and logical expressions.

In this chapter. we will look at a way of combining functions of smaller dimension into a larger space called **embedded landscapes**. These functions also produce functions of limited epistasis. I will perform Walsh analysis on these functions and discuss their properties. I will then show that two important function classes. NK-Landscapes and MAXSAT problems. have surprisingly similar mathematical structure and are both subclasses of embedded landscapes. Our knowledge about embedded landscapes will let us quickly deduce several properties of these function classes and point out how they might be different. In the process. I will develop the technique of using the Walsh distribution to discover the epistatic structure of a function.

6.1 Embedded Landscapes

An **Embedded landscape.** $f : \mathcal{B}^N \rightarrow \mathbb{R}$ can be expressed as the sum of P embedded functions:

$$f = \sum_{j=0}^{P-1} \mathcal{E}(g_j, m_j)$$

that is

$$f(x) = \sum_{j=0}^{P-1} g_j(pack(x, m_j))$$

Each $g_j : \mathcal{B}^{bc(m_j)} \rightarrow \mathbb{R}$ is an **interaction function** that weights the interaction captured by the bits selected by the **interaction mask.** $m_j \in \mathcal{B}^N$. There are no restrictions on the number of functions. P. or the number of 1 bits in each interaction mask. or the values returned by the interaction functions. The term embedded landscape comes from the idea that the g_j are often of lower dimension than the number of bits in the landscape. N. and are embedded in the higher dimensional space via function embedding.

Embedded landscapes allow us to independently control several important aspects of a function. By adjusting the value P. we control the number of subfunctions defined over different partitions of the search space. We can. for example. study functions with a variety of densities and distributions of subfunctions. With m_j. we can control the exact overlap between subfunction domains. For example. this could be used to study interaction and competitions between different partitions of the search space during genetic search. The subfunctions. g_j. control the ordering by fitness of hyperplanes in each partition defined by m_j. They also allow us to compare functions constructed out of simple basic classes of smaller subfunctions such as pit. random. linear and unimodal functions.

All of this can be done without invoking embedded landscapes. the model provides a ready-made conceptual and mathematical model with several useful orthogonal "knobs" for adjusting function structure. Furthermore. embedded landscape's simple weighted interaction model has the potential to represent many combinatorial problems that are the

sum of the effects of a limited number of interactions between objects. Such problems might include graph, flow, and network problems. As a test function generator, embedded landscapes provide the control and variety of important features needed for testing and analysis.

The history of embedded landscapes is short. Lee Altenberg (Altenberg, 1994; Altenberg, 1996) first introduced the idea of a generalized NK-landscape, a precursor to embedded landscapes, with a vector of interaction masks in the form of a matrix. His work only dealt with random interaction functions and was only intended as an extension of NK-landscapes. I independently created a generalization of NK-landscapes in (Heckendorn and Whitley, 1997) in which I referred to the concept as an NKP-landscapes. With NKP landscapes, I emphasized the generality of the embedded functions. Pointing out that the embedded functions need not necessarily be random functions. Since then my NKP-landscapes have been examined by other researchers (Smith and Smith, 1999). In (Heckendorn et al., 1999) I wanted to emphasize the embedding concept, so I have changed the name from NKP to embedded landscapes. I reluctantly chose to use the word "landscape" to show an association with NK-landscapes even though the connectivity of the domain is not specified.

6.2 An Analysis of Embedded Landscapes

An embedded landscape is a sum of simpler lower dimensional subfunctions. This method of combining subfunctions simplifies the analysis of embedded landscapes. To understand the structure of embedded landscapes, we must first understand how each lower dimensional subfunction is embedded in the higher dimensional landscape.

Let's briefly revisit function embedding for a single function in the context of an embedded landscape. Consider a subfunction $g : \mathcal{B}^M \rightarrow \mathbb{R}$ from an embedded landscape in the space $\mathcal{B}^L$. Assume the function uses mask m to select the interaction bits. $g(pack(x, m))$ is now a function in $\mathcal{B}^L$ in which all of the elements in hyperplane $h_{m,pack(x,m)}$ are set to $g(pack(x, m))$. Arranging the function values in a matrix by hyperplane, we get a matrix of function values with constant values for the elements in each column.

$$
\begin{array}{c}
\multicolumn{6}{c}{\longleftarrow\ pack(x,m)\ \longrightarrow} \\
\uparrow \\
pack(x,\overline{m}) \\
\downarrow
\end{array}
\begin{bmatrix}
g(0) & g(1) & g(2) & \ldots & g(2^{bc(m)}-1) \\
g(0) & g(1) & g(2) & \ldots & g(2^{bc(m)}-1) \\
\vdots & \vdots & \vdots & & \vdots \\
g(0) & g(1) & g(2) & \ldots & g(2^{bc(m)}-1)
\end{bmatrix}
\tag{6.1}
$$

Note that this matrix results from embedding just a *single* function g into f. From the matrix we can see that embedding a function using mask m induces a partitioning of the function domain and all the strings in each hyperplane of this partitioning of f are assigned a single function value as their fitness. This can be expressed simply by an example. If g uses mask 0001110000, then strings in f that are members of hyperplane ***000**** all have the same evaluation; all the strings in hyperplane ***001**** also all have the same evaluation; etc. Thus, a **plateau phenomenon** results in which flat regions, corresponding to specific hyperplanes, are present in the function. I will address this feature further in the analysis of this class of functions.

One of the major uses of embedded landscapes is to create functions by combining a number of smaller and often simpler subfunctions. Embedded landscapes are sufficiently general that any function can be expressed as an embedded landscape. However, if P becomes large in comparison with N, the interactions between the embedded functions become difficult to study. Furthermore, if $\max_{j=0..P-1}(bc(m_j))$ approaches N, then the interaction functions themselves are not much easier to study than the whole function itself.

6.2.1 Plateau Phenomena

When multiple subfunctions are used, the embedded landscape is the sum of these subfunctions. Each mask repartitions the space and each subfunction assigns values for each induced hyperplane of the subfunction. Yet flat regions can remain. Features of embedded

landscapes that tend to preserve plateaus are: small masks, overlapping masks, few subfunctions, subfunctions that themselves contain plateaus, subfunctions that have a small set of range values shared by all the subfunctions.

Formally, **plateaus** are a connected set of points in the domain that all have equal fitness. By **connected** I mean connected in the graph theoretic sense: we assume points are nodes and two points have an edge between them if they are Hamming distance one apart. Plateaus can, in fact, be defined for any neighborhood operator. It is best however, if the neighborhood represents paths that would be frequently chosen by the recombination/mutation operators of the algorithm of interest. Plateaus, as we have defined them, can cause difficulty for stochastic optimization algorithms that strongly rely on moves in Hamming space. This is because for points in the interior of the plateau and many along the edges, there is no productive fitness selection information. It is essentially an informationless surface.

6.2.2 Walsh Analysis

We know from the Embedding Theorem that if a single function $g : \mathcal{B}^M \rightarrow \Re$ is embedded in a higher dimension space, $\mathcal{B}^L$, $L > M$, using mask m, then the resulting function $f : \mathcal{B}^L \rightarrow \Re$ has Walsh coefficients:

$$w_i^f = \begin{cases} w^g_{pack(i,m)} & \text{if} \quad i \subseteq m \\ 0 & \text{otherwise} \end{cases}$$

where w^g and w^f are Walsh coefficients for g and f respectively: $i, m \in \mathcal{B}^L, bc(m) = M$. The critical observation is that even though all of the function values in f may be nonzero, only those Walsh coefficients, w_i^f, with $i \subseteq m$ can be nonzero. That is, for the Walsh transformed function $\mathcal{W}(f)$, $h_{\overline{m},0}$ contains the only nonzero Walsh coefficients. This leads to a second important observation: embedding a lower dimensional function, such as g above, in a higher dimensional space, as with function f above, neither increases the number of nonzero Walsh coefficients nor the maximum level of epistasis.

In terms of the matrix of Walsh coefficients, applying the Walsh transform to the earlier matrix (eq. 6.1) for the embedded function g yields:

$$
\begin{array}{cc}
 & \longrightarrow\ pack(x,m)\ \longrightarrow \\
 & \left[\begin{array}{ccccc}
w_0 & w_1 & w_2 & \cdots & w_{2^{bc(m)}-1} \\
0 & 0 & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & & \vdots \\
0 & 0 & 0 & \cdots & 0 \\
0 & 0 & 0 & \cdots & 0
\end{array}\right] \\
\uparrow & \\
pack(x,\overline{m}) & \\
\downarrow &
\end{array}
$$

where the w_i are the Walsh coefficients for g. (It is critical to note that the indices shown here are taken from g: their indices in f depend on the mask m.) Notice that the only nonzero Walsh coefficients have an index whose 1 bits are contained entirely in the mask m. Therefore, there are at most $2^{bc(m)}$ nonzero Walsh coefficients. All the other Walsh coefficients are zero. This is reasonable since bits outside of mask m should have no effect on the value of the higher dimensional function. Note that for values of $bc(m) \ll L$, the ratio of nonzero Walsh coefficients to the total number available becomes exponentially small, which strongly constrains the complexity of the function.

We use this to illustrate what happens as the number of embedded functions increases. When $P = 1$, the landscape consists of the embedding of a single subfunction g by a mask m resulting in every value in hyperplane $h_{m,x}$ being assigned $g(x)$. In Walsh space this makes the Walsh coefficients in the hyperplane $h_{\overline{m},0}$ the *only* possible nonzero Walsh coefficients. As P increases, the additive property of Walsh coefficients, instilled by the linearity of the Walsh transform, means that successive nonzero hyperplanes $h_{\overline{m_b},0}$ are added in the Walsh space while in function space, successive layers of constant hyperplanes $h_{m,x}$ are added.

Plateaus are guaranteed to exist in embedded landscapes for small P and small to moderate size interaction masks. As we saw with a single embedded function ($P = 1$), the

space is partitioned into $2^{bc(m_1)}$ hyperplanes, each partition being a plateau of $2^{bc(\overline{m_1})}$ in size. A second ($P = 2$) embedded function *tends* to redivide the space into a new set of hyperplanes cutting each hyperplane into $2^{bc(m_2)}$ pieces. To be precise, two subfunctions with interaction masks m_1 and m_2 produce at most $2^{bc(m_1 \vee m_2)}$ plateaus each $2^{bc(\overline{m_1 \vee m_2})}$ strings in size. Depending on the value of the subfunctions, there may be fewer and larger plateaus. This extends in the obvious manner to larger numbers of subfunctions.

As P increases, the overlaying of partitions tends to redivide the space into smaller and smaller plateaus, exponentially increasing the number of plateaus by $2^{bc(m_i)}$. As we will see, this process is slowed in the case of embedded functions which themselves have plateaus.

The Walsh coefficients of an embedded landscape are easily calculated from the Walsh coefficients of the subfunctions.

Theorem 59 (Walsh Coefficients for an Embedded Landscape)

If $f : \mathcal{B}^L \rightarrow \mathbb{R}$ is an embedded landscape with P subfunctions, $g_i : \mathcal{B}^{bc(m_i)} \rightarrow \mathbb{R}$ and interaction masks $m_i \in \mathcal{B}^L$, then

$$w_j^f = \sum_{j \subseteq m_i} w^{g_i}_{pack(j,m_i)}$$

where $w_j^f = 0$ if $j \not\subseteq m_i$. $\quad \forall i \in \{0, 1, 2, \ldots, P-1\}$

Proof:

The theorem follows from The Sum of Functions Theorem and the identity given by the Embedding Theorem:

$$w_i^f = \begin{cases} w^g_{pack(i,m)} & \text{if} \quad i \subseteq m \\ 0 & \text{otherwise} \end{cases}$$

□

The theorem points out that the only nonzero Walsh coefficients are those where the index of the Walsh coefficient is entirely contained in at least some interaction mask for the landscape.

Because it will come up so often I define the function $maxdim(f)$ for embedded landscape f as:

$$maxdim(f) = \max_{i=0}^{P-1} dim(g_i)$$

where g_i are the P subfunctions of f

The next theorem gives one of the most important properties of an embedded landscape.

Theorem 60 (Polynomial Time Walsh Analysis)

Let f be an embedded landscape with P subfunctions. g_i each of which uses an interaction mask of m_i. If the $maxdim(f) \leq K$ where K is a fixed constant independent of the $dim(f)$ then all of the Walsh coefficients of f can be computed in polynomial time relative to $dim(f)$.

Proof:

It is easy to enumerate all of the nonzero Walsh coefficients of an embedded landscape. The i^{th} subfunction contributes to Walsh coefficients:

$$\{w^f_{unpack(0,\ m_i)},\ w^f_{unpack(1,\ m_i)},\ \ldots,\ w^f_{unpack(2^{bc(m_i)}-1,\ m_i)}\}$$

Each of the P subfunctions contributes to at most 2^K Walsh coefficients. The rest are zero. Therefore, there are at most $2^K P$ nonzero Walsh coefficients that need to be computed, where K and P are constants. Each of these can be computed in polynomial time via The Walsh Coefficients for an Embedded Landscape Theorem. Therefore, all of the Walsh coefficients can be computed in polynomial time.

It is clear that even though an embedding of a function f may be nonzero for every value in the domain, the Ω of the function is not increased by embedding the function. This concept is distilled in the following theorem.

Theorem 61 (Embedded Function Epistatic Limit)

Let $g : \mathcal{B}^M \rightarrow \mathcal{R}$ and $f : \mathcal{B}^L \rightarrow \mathcal{R}$. If $\mathcal{E}(g, m)$ then $\Omega(f) = \Omega(g)$

Proof:

Since there is a 1-1 correspondence between the nonzero Walsh coefficients of f and g: and since $bc(i) = bc(pack(i, m))$ for $i \subseteq m$, the sets of Walsh coefficients for any given number of interacting bits must be identical, and hence $\Omega(f) = \Omega(g)$.

□

The result for an embedded function can now be easily extended to an embedded landscape.

Theorem 62 (Embedded Landscape Limit)

If f is an embedded landscape with embedded subfunctions g_i then

$$\Omega(f) \leq \max_{i=0}^{P-1} \Omega(g_i)$$

Proof:

Let $g'_i = \mathcal{E}(g_i, m_i)$, $g'_i : \mathcal{B}^L \rightarrow \mathcal{R}$. We know from the Embedded Function Epistatic Limit Theorem that the $\Omega(g'_i) = \Omega(g_i)$. From this it is clear by the Function Sum Theorem that:

$$\Omega(f) \leq \max_{i=0}^{P-1} \Omega(g_i)$$

□

This theorem says that even though the embedded landscape was defined over L bits, the maximum epistasis of the function is no larger than that of its most highly epistatic subfunction. This, of course, doesn't mean that the function is just as easy as solving the subfunctions independently.

6.3 The Distribution of Walsh Coefficients

In this section, I first analyze how embedded functions and landscapes cover the space of Walsh coefficients. I then show how to calculate the coverage more exactly if the masks are known and discuss the limitations of this computation. I will develop an important technique of using the distribution of possible nonzero Walsh coefficients as a signature of the breadth of possible functions obtainable by a class of functions.

Knowing the maximum epistasis is not as informative as knowing the distribution of nonzero Walsh functions called the Walsh distribution. This is measured as a vector $\vec{\mathcal{K}}$ where $\mathcal{K}_i$, the i^{th} Walsh count, is a count of the number of nonzero Walsh coefficients of order i.

Theorem 63 (Walsh Distribution for an Embedded Function)

Let $f = \mathcal{E}(g, m)$ where $f : \mathcal{B}^L \rightarrow \mathbb{R}$ and $g : \mathcal{B}^M \rightarrow \mathbb{R}$

$$\mathcal{K}_R \leq \binom{M}{R}$$

Proof:

Let $f = \mathcal{E}(g, m)$. We know from the Embedding Theorem that Walsh coefficients of f are the same as those of the subfunction g. We also know from our earlier discussions that a Walsh distribution of a function with all possible Walsh coefficients nonzero, forms a binomial distribution. If all the Walsh coefficients of g are nonzero we know that the distribution of the nonzero Walsh coefficients in f is also a binomial distribution and hence each $\mathcal{K}_R$ must be bounded by $\binom{M}{R}$.

Note that this theorem doesn't say that the limit is $\binom{\Omega(f)}{R}$) since $\Omega(f)$ does not limit the potential number of nonzero Walsh coefficients of any order less than or equal to $\Omega(f)$.

The following corollary suggests that because the distribution of the embedded function is binomial and the tails are limited by M and not N the number of potentially nonzero Walsh coefficients can be very small. We'll see this more dramatically later.

Corollary 63a (Embedded Function Sparseness)

Let $f = \mathcal{E}(g, m)$ where $f : \mathcal{B}^L \rightarrow \mathbb{R}$ and $g : \mathcal{B}^M \rightarrow \mathbb{R}$. There is at most one nonzero Walsh coefficient of order M and that is w_m.

Proof:

That there is only one follows immediately from the binomial distribution. That it is w_m follows from the Embedding Theorem.

Note that as with the earlier theorem, this does not say that there is necessarily only one nonzero Walsh coefficient with the highest epistasis.

Theorem 64 (Embedded Landscape Sparseness)

If f is an embedded landscape with P embedded subfunctions g_i then there can be no more than P Walsh coefficients of order $maxdim(f)$.

Proof:

This follows immediately from Embedded Function Sparseness.

□

Suppose we wish to model a landscape as an embedded function. If the embedded landscape is defined over $\mathcal{B}^L$, then there are $\binom{N}{maxdim(f)}$ different Walsh coefficients of order $maxdim(f)$. Therefore, by the Embedded Landscape Sparseness Theorem, one would need $\binom{N}{maxdim(f)}$ subfunctions to allow each of the Walsh coefficients to be nonzero. This means for an embedded landscape to model any function of limited epistasis would require potentially very many subfunctions or that the subfunctions be of limited epistasis as well.

Our reasoning about embedded landscape sparseness shows that the Walsh distribution of a function tells us a lot about its structure. If common functions tend to be sparse in Walsh space, then a compact way of expressing where the nonzeroness occurs, such as a Walsh distribution, may provide a convenient signature of functional structure. The signature may allow us to quickly decide when two functions are different in structure in some fundamental way.

Theorem 65 (Walsh Distribution Limit for Embedded Landscapes)

If f is an N bit embedded landscape with P embedded subfunctions g_i, then

$$\mathcal{K}_R \leq \min\left(\binom{N}{R}, \sum_{i=0}^{P-1}\binom{\dim(g_i)}{R}\right) \leq \min\left(\binom{N}{R}, P\binom{maxdim(f)}{R}\right) \qquad (6.2)$$

Proof:

The first upper bound is true because the number of nonzero Walsh coefficients can be overestimated by summing the number of Walsh coefficients for the subfunctions as if they were completely independent. The resulting value must be clipped by a min function to

the total number of possible nonzero Walsh coefficients for a N bit function. The second upper bound is true because each subfunction could be more coarsely bounded by the most epistatic subfunction in the landscape.

In practice. there is a lot of duplication when embedding masks overlap. The result is that the number of nonzero Walsh coefficients of order R is considerably less than our proposed upper bounds. In many circumstances where embedded landscapes are used as test functions the size of the subfunctions is constant. that is

$$dim(g_i) = maxdim(f) \qquad \forall i \in \{0.1.2.\ldots.P-1\}$$

I will call these a **fixed size embedded landscapes**.

For fixed size embedded landscapes the second upper bound applies with equal accuracy as the first upper bound. Fixed size embedded landscapes are just as general as nonfixed size embedded landscapes if the subfunctions are allowed to be functions of limited epistasis. Fixing the size at a value K gives us a single constant to talk about the size of the function. rather than considering a vector of the dimensions of all of the subfunctions. For simplicity. in much of the remainder of this chapter we will consider embedded landscapes to be of fixed size.

Table 6.1: The Upper Bound of Number of Nonzero Walsh Coefficients in a General 10 Bit Function vs the Number in a 10 Bit Fixed Embedded Landscape with 10 Subfunctions Broken Down By Order R and $\Omega = K$. When a Ratio is Given, the Numerator is the Upper Bound on the Number of Nonzero Walsh Coefficients for the Embedded Landscape and the Denominator is the Maximum Possible for any Function of Order N.

K— $\downarrow R$	1	2	3	4	5	6	7	8	9	10
0	1	1	1	1	1	1	1	1	1	1
1	10	10	10	10	10	10	10	10	10	10
2	-	10/45	30/45	45	45	45	45	45	45	45
3	-	-	10/120	40/120	100/120	120	120	120	120	120
4	-	-	-	10/210	50/210	150/210	210	210	210	210
5	-	-	-	-	10/252	60/252	210/252	252	252	252
6	-	-	-	-	-	10/210	70/210	210	210	210
7	-	-	-	-	-	-	10/120	80/120	120	120
8	-	-	-	-	-	-	-	10/45	45	45
9	-	-	-	-	-	-	-	-	10	10
10	-	-	-	-	-	-	-	-	-	1

Consider a fixed size embedded landscape f with $maxdim(f) = K$. The ratio of the maximum possible number of nonzero Walsh coefficients of order R in the landscape, $P\binom{K}{R}$, to the total number of possible order R Walsh coefficients, $\binom{N}{R}$ is a good indicator of sparseness of the coverage by embedded landscapes. Table 6.1 shows this comparison for an 10 bit embedded landscape of 10 embedded functions. The columns represent the varying values of K. The rows represent the various orders, R, of Walsh coefficients for the function. Therefore each column represents the Walsh distribution for a different embedded landscape. Wherever a single number occurs in the table, the embedded landscape may have the maximum number of nonzero Walsh coefficients for that order. When a ratio is given, the numerator is the upper bound on the number of nonzero Walsh coefficients for the embedded landscape and the denominator is the maximum possible for any function of order N. For instance, for an embedded landscape with $N = 10$ and $K = 5$ there are $\binom{10}{5} = 252$ Walsh coefficients of order 5 but at most $10\binom{6}{5} = 60$ of them can be nonzero in the embedded landscape.

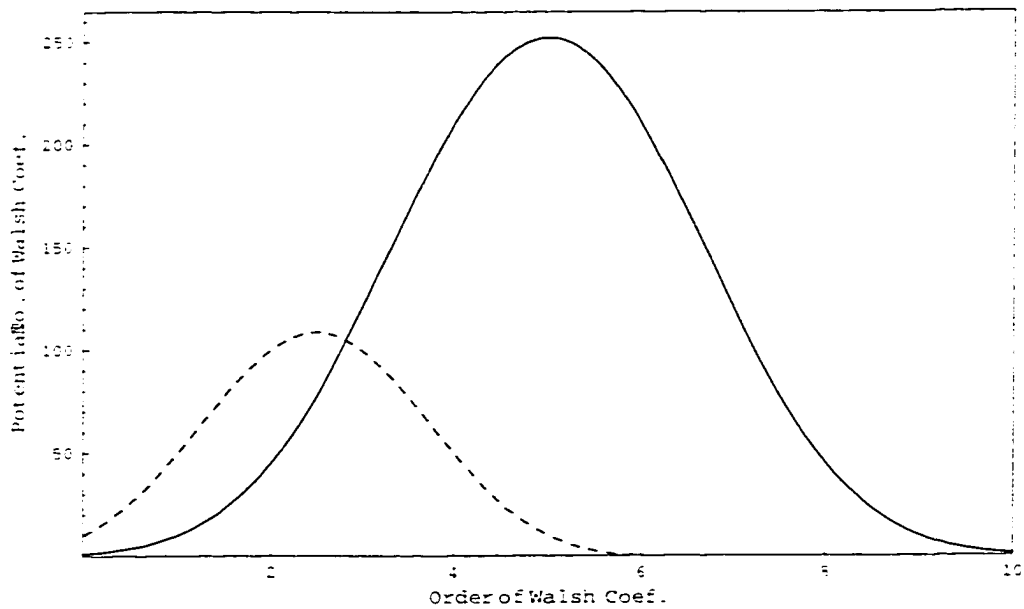


Figure 6.1: Walsh Coefficient Distributions for an Arbitrary 10 Bit Function (Solid Line) and for a 10 Bit Fixed Embedded Landscape (Dashed Line) with 10 Subfunctions Each with $\Omega = 5$

Another way to look at this 10 bit embedded landscape is presented in Figure 6.1. Here the x-axis represents the order, R, of the Walsh coefficients. The y-axis represents the num-

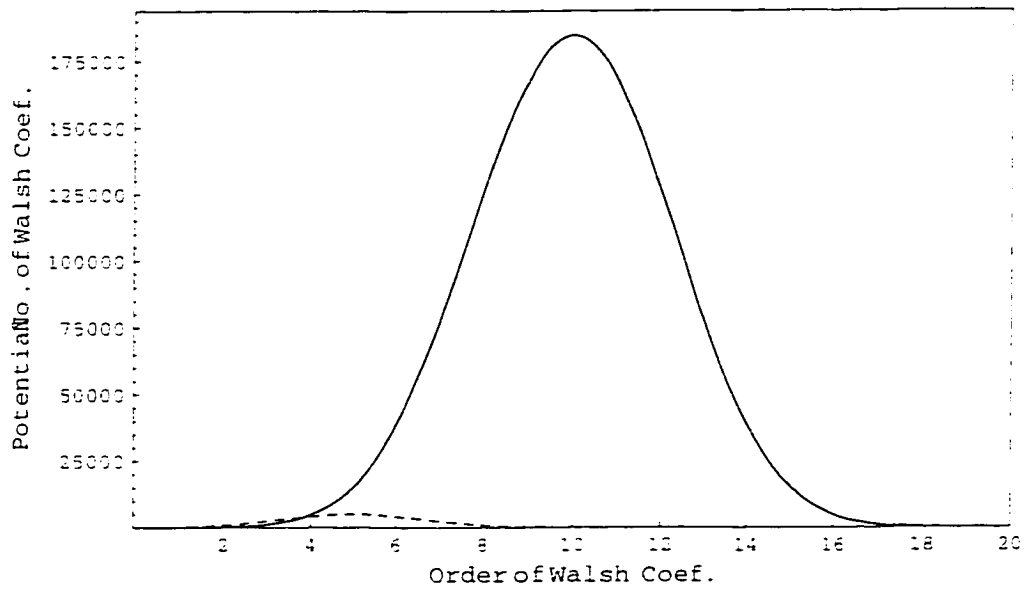


Figure 6.2: Walsh Coefficient Distributions for an Arbitrary 20 Bit Function (Solid Line) and for a 20 Bit Fixed Embedded Landscape (Dashed Line) with 20 Subfunctions Each with $\Omega = 10$

ber of possible Walsh coefficients of that order. K, which is also the Ω of the subfunctions, is 5. The **solid line** represents the maximum number of possible nonzero Walsh coefficients for an arbitrary function over the 10 bit domain. The standard continuous Gaussian approximation for a binomial distribution is used to ease visualization (Mendenhall, 1967). The **dashed line** shows the maximum number of nonzero Walsh coefficients for a 10 bit embedded landscape with 10 subfunctions and the Ω, for each subfunction of 5. This graph shows that the greatest number of possible Walsh coefficients are of order $N/2$ or 5. When the 10 functions are summed, the coverage is increased 10 times and the left tail of the upper dashed curve arches above the solid curve. It is the minimum of these two curves, as per Eq. 6.2, that limits the total number of Walsh coefficients for the embedded landscape.

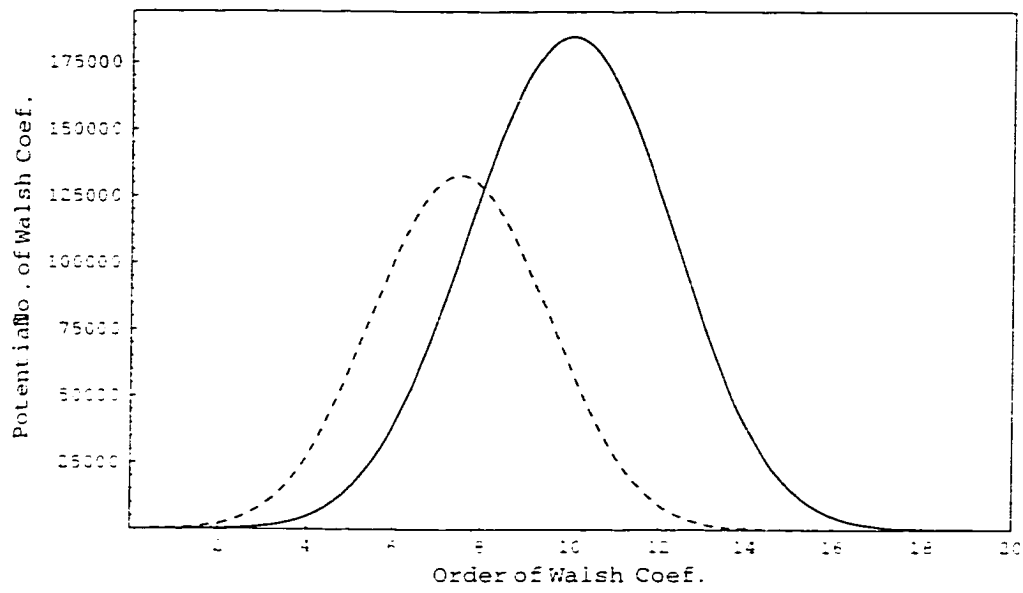


Figure 6.3: Walsh Coefficient Distributions for an Arbitrary 20 Bit Function (Solid Line) and for a 20 Bit Fixed Embedded Landscape (Dashed Line) with 20 Subfunctions Each with $\Omega = 15$

Table 6.2: The Upper bound of the Coverage and Intersection Points for embedded landscapes

P=N=40			P=N=80		
K	Coverage	I	K	Coverage	I
0	$1.00E+00$	2	0	$1.00E+00$	2
4	$1.42E-03$	2	8	$1.52E-07$	2
8	$5.21E-05$	3	16	$7.65E-11$	3
12	$1.53E-05$	4	24	$4.22E-12$	4
16	$2.21E-05$	5	32	$7.81E-12$	5
20	$1.11E-04$	6	40	$2.31E-10$	7
24	$1.27E-03$	8	48	$3.79E-08$	9
28	$1.93E-02$	10	56	$9.54E-06$	12
32	$2.76E-01$	16	64	$2.44E-03$	19
36	$9.98E-01$	28	72	$5.27E-01$	36
39	$1.00E+00$	–	79	$1.00E+00$	–

As N increases, the discrepancy between embedded landscapes and arbitrary functions of $\Omega = K$ becomes increasingly pronounced. In Figure 6.2, the solid curve is the potential number of nonzero Walsh coefficients for a 20 bit function, while the small dashed curve at the bottom of the graph is the distribution for an embedded landscape of 20 functions with $\Omega = 10$. In Figure 6.3, the dashed curve is the distribution for an embedded landscape of 20 functions with Ω now set to 15. This illustrates that as Ω increases and the mean of the distribution of nonzero Walsh coefficients moves to the right. When $\Omega = N$ all possible functions of order N are covered.

In Table 6.2, we give the intersection points and coverage for embedded landscapes of $N = 40$ and $N = 80$ for various values of K and the number of subfunctions P equal to N. The **coverage** is the ratio of the number of possible nonzero Walsh coefficients to available Walsh coefficients. The **intersection point**, I, is the intersection of the maximum possible coverage of N independent K order functions and the limit imposed by the general N bit function. I can be found at the intersection of upper dashed line and the solid line on the graph, as measured in bits of interaction. Note that I remains small for most values of K and only nears N when K nears N. This allows the number of nonzero Walsh coefficients for an embedded landscape to be approximated by $N2^{K-1}$ for most values of K except near 0 and N. The number of available Walsh coefficients of a given order or less is approximately 2^{N-1} for $K \geq N/2$. Therefore, for $K \geq N/2$ and not near N, an upper bound for the coverage is $4N/2^{N-K}$. As K decreases from N/2, the number of coefficients for an embedded landscape continues to decrease by powers of two while the number of available coefficients initially decreases more slowly. Thus, for most $K < N/2$, where K is not near 0, the value of the coverage is less than it is at $K = N/2$ giving an upper bound of $4N/2^{N/2}$.

6.3.1 Computing the Walsh Distribution by Masks

In the previous section we showed an upper bound for the distribution of nonzero Walsh coefficients. This assumed that none of the N bit positions has a 1 in more than one of the interaction masks m_b. If we know the set of masks, we can better predict the upper bound

of the distribution of Walsh coefficients. This gives us some insight on how overlapping regions of bit interaction interfere to reduce the diversity of functions as seen by the Walsh coefficients.

We know from the Embedding Theorem that the nonzero Walsh coefficients for the embedding of a function must have indices that are contained in the mask used for the embedding. That is, if $f = \mathcal{E}(v, m)$ then $w_i^f \neq 0$ only if $i \subseteq m$. Each interaction mask can therefore generate 2^{K-1} nonzero Walsh coefficients in the final function, but only if there is not any overlapping between masks.

But suppose there is overlapping. Let's examine the case of a function f that is composed of the sum of the embeddings of two functions, f_1 and f_2, based on corresponding overlapping masks m_1 and m_2 with $bc(m_1) = \delta_1$ and $bc(m_2) = \delta_2$. Assume the masks share δ_{12} bits in common, i.e. $bc(m_1 \wedge m_2) = \delta_{12}$. The maximum number of nonzero Walsh coefficients for function f is the sum of the coefficients covered by the two functions minus the coefficients duplicated by the intersection.

$$2^{\delta_1} + 2^{\delta_2} - 2^{\delta_{12}}$$

In fact, because of the 1-1 correspondence established in the corollary to the Embedding Theorem, we know that the Walsh counts $\mathcal{K}_i$ for f_1 and f_2 for a given i can only influence the order i Walsh count for function f. Therefore the above upper bound can be stated more strongly as:

$$\mathcal{K}_i^f \leq \mathcal{K}_i^1 + \mathcal{K}_i^2 - \mathcal{K}_i^{12}$$

where $\mathcal{K}_i^1$ is the i^{th} Walsh count for the function generated by an embedding based on m_1 and $\mathcal{K}_i^{12}$ is the Walsh count for a function generated by an embedding based on $m_1 \wedge m_2$.

For example: In Table 6.3 we assume the function f is composed of the embedding of two functions f_1 and f_2 on the domain $\mathcal{B}^4$ expanded with the respective masks $m_1 =$ 1110001 and $m_2 =$ 0011101. The first column is the order of the Walsh coefficients being counted. Since the masks are 7 bits long, the domain of f is $\mathcal{B}^7$, and therefore, there are 8 different

Table 6.3: Example Walsh Count Computation

i	K_i^1	K_i^2	K_i^{12}	K_i^f
0	1	1	1	1
1	4	4	2	6
2	6	6	1	11
3	4	4	0	4
4	1	1	0	1
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0

Walsh counts for f and the embeddings of f_1 and f_2. The second column is the distribution of the maximum number of nonzero Walsh coefficients for the embedding of f_1 given that the embedding was done using a mask with 4 bits set. f_2 is similarly represented in column three. The fourth column is the maximum number of nonzero Walsh coefficients that are **duplicated** in the interaction of the two masks. The duplicate counts are subtracted and yield the final column which is the maximum number of nonzero coefficients for the given masks.

In general, by using the Inclusion-Exclusion Principle (Niven, 1965) from combinatorics we can see for f based on N functions that are expanded using masks $m_b : b \in \{1, 2, \ldots, N\}$ the distribution of Walsh counts can be calculated as follows:

$$\vec{\mathcal{K}}^f = \sum_{1 \le i \le N} \vec{\mathcal{K}}^i - \sum_{1 \le i < j \le N} \vec{\mathcal{K}}^{ij} + \sum_{1 \le i < j < k \le N} \vec{\mathcal{K}}^{ijk} - \cdots - (-1)^N \vec{\mathcal{K}}^{(allmasks)}$$

The first term in the expression represents the sum of the contributions of each mask if the masks were all disjoint. Each successive sum in the expression is an interaction term for increasing numbers of masks.

Why doesn't this provide a formula for the exact number of nonzero Walsh coefficients? The answer is best illustrated by going back to the two mask example. The Walsh counts do not contain all of the information necessary to determine which Walsh coefficients are covered by both f_1 and f_2 when all possible Walsh coefficients of each order are **not** covered. Of course, an exhaustive listing of which coefficients are covered and which are not by each embedding function will determine the exact coverage.

6.3.2 Summary Statistics for Embedded Landscapes

Walsh analysis can be used to compute the **summary statistics** (mean, variance, skew, ...) for a function. Clearly, computing summary statistics for arbitrary fitness functions would require exponential time relative to the size of the domain in bits. However, we have shown that the Walsh coefficients for an embedded landscape can be computed in polynomial time relative to the number of bits in the domain of the function and in polynomial time relative to the number of clauses. Finally, we showed that there were a polynomial number of

nonzero Walsh coefficients. Soraya Rana suggested that the r^{th} moments of the functions might also be computed in polynomial time and did some preliminary work on a proof.

In this section, we show that given the Walsh coefficients of a function it takes a polynomial time, as measured in the number of nonzero Walsh coefficients, to compute a variety of statistical moments. With that result, it is easy to show that given the Walsh coefficients of an embedded landscape, it is a polynomial time transformation to compute a variety of statistical moments. Goldberg and Rudnick (Goldberg and Rudnick, 1991) have used Walsh coefficients to calculate fitness variance for fitness functions and hyperplanes, but I exceed their formulations by computing moments of arbitrary degree about three different means.

Given the mean, the formula used to compute the r^{th} moment, denoted μ_r, for a discrete random variable X is:

$$\mu_r = E[(X-\mu)^r] = \sum_{x \in X} (x-\mu)^r p(x) \tag{6.3}$$

where $p(x)$ is the probability of x occurring. Given the r^{th} moment, it is easy to compute the variance, skew and kurtosis for any fitness function (Spiegel, 1961).

$$\text{variance} = \mu_2 = \sigma^2 \qquad \text{skew} = \frac{\mu_3}{\sigma^3} \qquad \text{kurtosis} = \frac{\mu_4}{\sigma^4}$$

There are three kinds of moment we will wish to calculate. First is the moment of the entire function about the mean of the entire function, denoted μ. The second is the moment of a hyperplane about the mean for the entire function. This tells us how the values in the hyperplane compare with the average for the whole function. Finally, we want the moment of the hyperplane about the mean of the hyperplane.

Theorem 66 (Moment about Function Mean)

The moment about the mean μ for function f given the Walsh coefficients of f is:

$$\mu_r = \sum_{a_1 \doteq a_2 \doteq \ldots \doteq a_r = 0} w_{a_1} w_{a_2} \ldots w_{a_r}. \qquad a_i \neq 0 \; \forall \; i$$

Proof:

For the r^{th} moment calculation. I assume each function value is equally likely. Therefore, by definition of moment in Equation 6.3: for $f : \mathcal{B}^L \rightarrow \Re$ and $p(x) = \frac{1}{2^L}$:

$$\mu_r = \sum_{x \in X} \frac{(x-\mu)^r}{2^L}$$

Since X represents a real valued function over an L bit domain then:

$$\mu_r = \frac{1}{2^L} \sum_{x=0}^{2^L-1} (f(x) - \mu)^r$$

We can then substitute for f with the Walsh representation of f:

$$\mu_r = \frac{1}{2^L} \sum_{x=0}^{2^L-1} \left(\sum_{i=0}^{2^L-1} w_i \psi_i(x) - \mu \right)^r$$

Since $\mu = w_0$, and $\psi_0(x) = 1 \ \forall x$:

$$\mu_r = \frac{1}{2^L} \sum_{x=0}^{2^L-1} \left(\sum_{i=1}^{2^L-1} w_i \psi_i(x) \right)^r$$

We can now expand the exponential creating a set of r indices a_j where $a_j \in \mathcal{B}^L$:

$$\mu_r = \frac{1}{2^L} \sum_{x=0}^{2^L-1} \left(\sum_{a_1=1}^{2^L-1} w_{a_1} \psi_{a_1}(x) \right) \left(\sum_{a_2=1}^{2^L-1} w_{a_2} \psi_{a_2}(x) \right) \cdots \left(\sum_{a_r=1}^{2^L-1} w_{a_r} \psi_{a_r}(x) \right)$$

Since the Walsh coefficients do not depend on x, the formula can be rewritten as:

$$\mu_r = \frac{1}{2^L} \sum_{a_1=1}^{2^L-1} \sum_{a_2=1}^{2^L-1} \cdots \sum_{a_r=1}^{2^L-1} w_{a_1} w_{a_2} \cdots w_{a_r} \sum_{x=0}^{2^L-1} \psi_{a_1}(x) \psi_{a_2}(x) \cdots \psi_{a_r}(x)$$

Using the fact that for arbitrary p and q: $\psi_p(x)\psi_q(x) = \psi_{p \dot{-} q}(x)$:

$$\mu_r = \frac{1}{2^L} \sum_{a_1=1}^{2^L-1} \sum_{a_2=1}^{2^L-1} \cdots \sum_{a_r=1}^{2^L-1} w_{a_1} w_{a_2} \cdots w_{a_r} \sum_{x=0}^{2^L-1} \psi_{a_1 \dot{-} a_2 \dot{-} \cdots \dot{-} a_r}(x)$$

Now using the Balanced Sum Theorem we see that **only when** $a_1 \oplus a_2 \oplus \ldots \oplus a_r = 0$ is the inner sum nonzero. Therefore.

$$\mu_r = \frac{1}{2^L} \sum_{a_1 \oplus a_2 \oplus \ldots \oplus a_r = 0} w_{a_1} w_{a_2} \ldots w_{a_r} 2^L. \qquad a_i \neq 0 \ \forall \ i$$

$$= \sum_{a_1 \oplus a_2 \oplus \ldots \oplus a_r = 0} w_{a_1} w_{a_2} \ldots w_{a_r}. \qquad a_i \neq 0 \ \forall \ i \tag{6.4}$$

□

To summarize. given the set of nonzero Walsh coefficients. we can compute the r^{th} moment for the fitness distribution using products of the Walsh coefficients such that the exclusive-or of the indices is zero.

This formula allows us to compute the variance. skew and kurtosis for any fitness distribution provided we are given the Walsh coefficients.

$$variance = \mu_2 = \sigma^2 \qquad skew = \frac{\mu_3}{\sigma^3} \qquad kurtosis = \frac{\mu_4}{\sigma^4}$$

For example. since $a_1 \oplus a_2 = 0$ if and only if $a_1 = a_2$ then the variance for any function can be computed

$$\sum_{i=1}^{2^L - 1} w_i w_i$$

This is the formula I computed in our discussion of dot products.

Of course. this computation of the moment around the mean. if done directly. would take $\mathcal{O}(2^{Lr})$ time. However. in the case of an embedded landscape. only a polynomial number of Walsh coefficients are nonzero. they are easily enumerated. and only the nonzero coefficients need be considered. Selecting the indices to have even parity would consist of selecting the first $r - 1$ indices from the set of nonzero Walsh coefficients. The exclusive-or of these would be taken and would be used as the desired r^{th} index. The exclusive-or of the r indices would therefore be zero. Using this simple strategy. it would take $\mathcal{O}(n^{r-1})$ time to compute the r^{th} moment given the Walsh coefficients. where n is the number of nonzero Walsh coefficients. In the case of embedded landscapes. the nonzero Walsh coefficients can

be identified and computed in polynomial time. Since our moment computation strategy is also polynomial time for fixed r, any summary statistics can be computed from Theorem 66 in polynomial time.

This same approach can be used to calculate summary statistics for hyperplanes. A subtle feature of the derivation of moments from Walsh coefficients is that w_0 was the mean for the **entire** function, not just a hyperplane. So there are really two obvious types of moments for a hyperplane: the moment about the mean for the entire function and the moment about the mean for just the hyperplane itself. We will treat these two cases in that order.

Theorem 67 (Moment of Hyperplane about Function Mean)

The r^{th} moment of the elements of hyperplane h about the mean μ for function f given the Walsh coefficients of f is:

$$\mu_r(h) = \sum_{a_1 \div a_2 \div \ldots \div a_r \subseteq \alpha(h)} w_{a_1} w_{a_2} \ldots w_{a_r} \psi_{a_1 \div a_2 \div \ldots \div a_r}(\beta(h)). \qquad a_i \neq 0 \ \forall \ i$$

Proof:

In the case of the moment of f in hyperplane h about the mean for the entire function:

$$\mu_r(h) = \frac{1}{|h|} \sum_{x \in h} (f(x) - \mu)^r$$

where μ is the mean for the entire function. We now proceed as with the earlier derivation:

$$\mu_r(h) = \frac{1}{|h|} \sum_{a_1=1}^{2^L-1} \sum_{a_2=1}^{2^L-1} \cdots \sum_{a_r=1}^{2^L-1} w_{a_1} w_{a_2} \ldots w_{a_r} \sum_{x \in h} \psi_{a_1 \div a_2 \div \ldots \div a_r}(x)$$

Then applying the Balanced Sum Theorem for Hyperplanes we get:

$$\mu_r(h) = \frac{1}{|h|} \sum_{a_1 \div a_2 \div \ldots \div a_r \subseteq \alpha(h)} w_{a_1} w_{a_2} \ldots w_{a_r} (\psi_{a_1 \div a_2 \div \ldots \div a_r}(\beta(h)) |h|). \qquad a_i \neq 0 \ \forall \ i$$

Therefore, the r^{th} moment about the mean for the entire function over hyperplane h is:

$$\mu_r(h) = \sum_{a_1 \div a_2 \div \ldots \div a_r \subseteq \alpha(h)} w_{a_1} w_{a_2} \ldots w_{a_r} \psi_{a_1 \div a_2 \div \ldots \div a_r}(\beta(h)). \qquad a_i \neq 0 \ \forall \ i$$

Now consider the case where the mean used in the moment calculations is the mean of the hyperplane. We denote this moment for hyperplane h as $\hat{\mu}_r(h)$.

Theorem 68 (Moment of Hyperplane about Hyperplane Mean)

The moment of the elements of hyperplane h about the mean μ for function f given the Walsh coefficients of f is:

$$\hat{\mu}_r(h) = \sum_{a_1 \div a_2 \div \ldots \div a_r \subseteq \alpha(h)} w_{a_1} w_{a_2} \ldots w_{a_r} \psi_{a_1 \div a_2 \div \ldots \div a_r}(\beta(h)). \qquad a_i \not\subseteq \alpha(h) \ \forall \ i$$

Proof:

We refer to the mean of the hyperplane as $\hat{\mu}$. Returning to the original equation for moment we get:

$$\hat{\mu}_r(h) = \frac{1}{|h|} \sum_{x \in h} (f(x) - \hat{\mu})^r$$

Substituting the Walsh transform for the function f and the results of the Hyperplane Averaging theorem for $\hat{\mu}$ we get:

$$\hat{\mu}_r(h) = \frac{1}{|h|} \sum_{x \in h} \left(\sum_{i=0}^{2^L - 1} w_i \psi_i(x) - \sum_{k \subseteq \alpha(h)} w_k \psi_k(\beta(h)) \right)^r$$

The left sum in parentheses can now be broken into two parts

$$\hat{\mu}_r(h) = \frac{1}{|h|} \sum_{x \in h} \left(\sum_{i \subseteq \alpha(h)} w_i \psi_i(x) + \sum_{j \not\subseteq \alpha(h)} w_j \psi_j(x) - \sum_{k \subseteq \alpha(h)} w_k \psi_k(\beta(h)) \right)^r$$

Regrouping under the sums gives

$$\hat{\mu}_r(h) = \frac{1}{|h|}\sum_{x\in h}\Big(\sum_{i\subseteq\alpha(h)}\big(w_i\psi_i(x) - w_i\psi_i(\beta(h))\big) + \sum_{j\subseteq\alpha(h)} w_j\psi_j(x)\Big)^r$$

Note that $x \in h$ and $i \subseteq \alpha(h)$ therefore. $\psi_i(x) = \psi_i(\beta(h))$! This means $w_i\psi_i(x) - w_i\psi_i(\beta(h))$ is zero and we get

$$\hat{\mu}_r(h) = \frac{1}{|h|}\sum_{x\in h}\Big(\sum_{j\subseteq\alpha(h)} w_j\psi_j(x)\Big)^r$$

Proceeding with the expansion of the r^{th} power as we did in the earlier proofs:

$$\begin{aligned}
\hat{\mu}_r(h) &= \frac{1}{|h|}\sum_{x\in h}\Big(\sum_{a_1\subseteq\alpha(h)} w_{a_1}\psi_{a_1}(x)\Big)\Big(\sum_{a_2\subseteq\alpha(h)} w_{a_2}\psi_{a_2}(x)\Big)\cdots\Big(\sum_{a_r\subseteq\alpha(h)} w_{a_r}\psi_{a_r}(x)\Big)\\
&= \frac{1}{|h|}\sum_{x\in h}\sum_{a_1\subseteq\alpha(h)}\sum_{a_2\subseteq\alpha(h)}\cdots\sum_{a_r\subseteq\alpha(h)} w_{a_1}w_{a_2}\ldots w_{a_r}\psi_{a_1\dot{+}a_2\dot{+}\ldots\dot{+}a_r}(x)\\
&= \frac{1}{|h|}\sum_{a_1\subseteq\alpha(h)}\sum_{a_2\subseteq\alpha(h)}\cdots\sum_{a_r\subseteq\alpha(h)} w_{a_1}w_{a_2}\ldots w_{a_r}\sum_{x\in h}\psi_{a_1\dot{+}a_2\dot{+}\ldots\dot{+}a_r}(x)\\
&= \frac{1}{|h|}\sum_{a_1\subseteq\alpha(h)}\sum_{a_2\subseteq\alpha(h)}\cdots\sum_{a_r\subseteq\alpha(h)} w_{a_1}w_{a_2}\ldots w_{a_r}|h|\psi_{a_1\dot{+}a_2\dot{+}\ldots\dot{+}a_r}(\beta(h)).\\
&\qquad\qquad \text{with } a_1 \ominus a_2 \ominus \ldots \ominus a_r \subseteq \alpha(h)\\
&= \sum_{a_1\subseteq\alpha(h)}\sum_{a_2\subseteq\alpha(h)}\cdots\sum_{a_r\subseteq\alpha(h)} w_{a_1}w_{a_2}\ldots w_{a_r}\psi_{a_1\dot{+}a_2\dot{+}\ldots\dot{+}a_r}(\beta(h)).\\
&\qquad\qquad \text{with } a_1 \ominus a_2 \ominus \ldots \ominus a_r \subseteq \alpha(h)
\end{aligned}$$

Which is the same as saying:

$$\hat{\mu}_r(h) = \sum_{a_1\dot{+}a_2\dot{+}\ldots\dot{+}a_r\subseteq\alpha(h)} w_{a_1}w_{a_2}\ldots w_{a_r}\psi_{a_1\dot{+}a_2\dot{+}\ldots\dot{+}a_r}(\beta(h)). \qquad a_i \not\subseteq \alpha(h)\ \forall\ i$$

□

Note that when h is fixed as all $*$'s. that is. h is the whole domain then, $\alpha(h) = 0$ and the Moment of Hyperplane about Hyperplane Mean Theorem becomes the same as the first theorem in this section.

6.3.3 Embedded Landscapes and the Central Limit Theorem

Embedded landscapes take on the appearance of a sum of distributions. It would seem that the Central Limit Theorem should apply to them. In this section I present a conjecture that I have yet to prove and yet is an important contribution to understanding the statistical nature of embedded landscapes. Mendenhall (Mendenhall. 1967) presents the **Central Limit Theorem** theorem as:

> *If random samples of n observations are drawn from a population with finite mean, μ, and standard deviation, σ, then, when n is large, the sample mean, $\overline{y}$, will be approximately normally distributed with mean equal to μ and standard deviation $\sigma/\sqrt{n}$. The approximation will become more and more accurate as n becomes large.*

By "random samples" he means drawing each element from a finite population with equal probability. Mendenhall notes that the theorem can be restated in terms of the sum of sample measures, $\sum_{i=1}^{n} y_i$. In this case the mean would tend to $n\mu$ and the standard deviation would tend to $\sqrt{n}\sigma$. An obvious extension to this would be the sample sum can be made from n possibly *distinct* populations so that, for example, y_1 is draw from population p_1 and y_2 is drawn from population p_2, etc. If each population p_i represents a subfunction in an embedded landscape, then the function value for the landscape represents the sum of samples, $\sum_{i=1}^{n} y_i$. In this case the sample distribution will be normally distributed with a mean equal to $\sum_{i=1}^{n} \mu_i$, where μ_i is the mean of the i^{th} population. There is one important proviso. The domains of the subfunctions must be independent. If they are not, then selecting a value for a subfunction biases which members of overlapping subfunctions can be chosen, that is, the subfunctions are not chosen with equal likelihood.

I propose the **Central Limit Conjecture for Embedded Landscapes** that the central limit theorem for sums of samples still applies to embedded functions even though the functions are no longer truly randomly sampled. If this conjecture is true, then it is a general property of embedded landscapes that as the number of subfunctions increase the values for an embedded landscape tend to cluster about the mean with a predictable standard deviation.

6.4 The Walsh Analysis of Two Classic Problems

Test function generators have become a new standard in testing the performance of evolutionary algorithms (Whitley et al., 1995b; Jong et al., 1997; Heckendorn et al., 1999). Performing comparative studies between search algorithms is complicated by the fact that there are few well-understood test problems available to researchers. Often a small test suite of parameter optimization problems was used to determine algorithm performance. Using this small seemingly arbitrary set of test functions provided a very narrow view of performance. Test function generators, on the other hand, allow researchers to have a virtually limitless number of test problems that all fall within a broad class of functions.

Two well known examples of classes of test function generators are MAXSAT problems and NK-landscapes. MAXSAT problems and NK-landscapes can easily be made into test function generators because the problem definitions are general and simple. Both problems have tunable parameters that allow the user to manipulate the character of the problems. Since the domain for both MAXSAT and NK-landscape functions is the space of binary strings, the representation issues and choice of genetic operators are simplified. These two domains appear, superficially, to be ideal for testing the performance and behavior of genetic algorithms (Jong and Spears, 1989; Jong et al., 1997). But are they?

6.4.1 NK-landscapes Defined

NK-landscapes (Kauffman, 1993) are a popular experimental model for correlated landscapes (Weinberger, 1990) taken from the realm of theoretical biology. An NK-landscape is a function $f : \mathcal{B}^N \rightarrow [0,1)$ where K is the number of bits in the string that epistatically interact with each bit. An NK-landscape can be expressed as an average of N functions as follows:

$$f(x) = \frac{1}{N} \sum_{j=0}^{N-1} \tau_j(pack(x, m_j))$$

Each $\tau_j : \mathcal{B}^{K+1} \rightarrow [0,1)$ is an evaluation function which gives a partial fitness for each bit pattern formed by the value of the j^{th} bit itself and the K bits with which it interacts.

Each r_j can be implemented as a table of 2^{K+1} random real values in the rand $[0.1)$. Each bit may interact with a possibly *different* set of K bits. To get this behavior, N masks, $m_j : bc(m_j) = K + 1$, are used to select the K bits that epistatically interact with the j^{th} bit. The j^{th} bit itself is also selected by the mask. Therefore, $bc(m_j) = K + 1$ and K must fall in the range $[0. N - 1]$. The *pack* function uses the masks to select the interacting bits and generate the arguments for the interaction functions r_j.

One of the nice features of NK-landscapes is that K acts as a tunable ruggedness control. When $K = 0$, a bitwise linear function is generated. The resulting landscape is the average of the weights associated with each bit and hence, assuming a Hamming neighborhood, is highly correlated and relatively smooth. When $K = N - 1$ the function is random and uncorrelated. Note that NK-landscapes, like embedded landscapes, are technically not landscapes but rather functions, since no neighborhood description has been specified for the domain elements (Jones, 1995).

6.4.2 MAXSAT Problems Defined

The second classic type of problem is MAXSAT which is based on the **k-satisfiability** or kSAT problem (Hogg et al., 1996). MAXSAT problems have been used as a testbed for GAs (Jong et al., 1997). We define the MAXSAT function $f : \mathcal{B}^N \rightarrow \mathbb{R}$, as we did earlier:

$$f(x) = \sum_{j=0}^{c-1} c_j(pack(x, m_j))$$

where c is the number of disjunctive clauses in the CNF and where $x \in \mathcal{B}^N$ represents the TRUE/FALSE assignments to each of the N Boolean variables in the problem. Each c_j represents a Boolean evaluation function applied to the disjunctive clause that uses the k variables selected by mask $m_j : bc(m_j) = k$ using the *pack* function. In this case $c_j : \mathcal{B}^k \rightarrow \mathcal{B}$.

6.4.3 MAXSAT Problems and NK-landscapes as Embedded Landscapes

It is apparent from their definitions that both NK-landscapes and MAXSAT problems are forms of embedded landscapes. Let's look at an example of how a MAXSAT problem and an NK-landscape can both be expressed as an embedded landscape. Let $v = [A, B, C, D]$

be the vector of variables and vector $x \in \mathcal{B}^4$ be an assignment of Boolean values to the variables. Consider the following the MAX2SAT problem:

$$(A \vee \overline{B}) + (B \vee \overline{C}) + (\overline{A} \vee C) + (\overline{B} \vee \overline{D})$$

and an NK-landscape with $N = 4$ and $K = 1$ with the following bit interactions:

$$m_0 = 1100, m_1 = 0110, m_2 = 1010, m_3 = 0101$$

and associated epistatic functions r_0, r_1, r_2, r_3 to be described below. Both these functions can be represented as embedded landscapes. For convenience, the functions are chosen so they have the same interaction masks. The interaction functions for both problems are defined in Table 6.4.

As an embedded landscape, the MAX2SAT has each disjunctive clause represented as an interaction function c_i and the value of each logical variable as a bit position in the argument string. Then the MAX2SAT problem SAT : $\mathcal{B}^4 \rightarrow \mathcal{B}$ is

$$\mathrm{SAT}(x) = c_0(pack(x, m_0)) + c_1(pack(x, m_1)) + c_2(pack(x, m_2)) + c_3(pack(x, m_3))$$

The NK-landscape NK : $\mathcal{B}^4 \rightarrow \mathbb{R}$ is

$$\mathrm{NK}(x) = \frac{1}{4}(r_0(pack(x, m_0)) + r_1(pack(x, m_1)) + r_2(pack(x, m_2)) + r_3(pack(x, m_3)))$$

with the interaction functions as shown in table 6.4.

6.5 An Analysis of MAXSAT Problems

Since a MAXSAT problem is an embedded landscape, the maximum level of epistasis, Ω, is the maximum number of variables in any clause. To compute the Walsh coefficients for the MAXSAT problem, we need to first compute the Walsh coefficients for the subfunctions.

Table 6.4: Values of the Subfunctions for both NK-Landscapes and MAXSAT Example Problems

	NK subfunctions				2SAT subfunctions			
$pack(x, m_i)$	r_0	r_1	r_2	r_3	c_0	c_1	c_2	c_3
00	0.31	0.41	0.59	0.26	1	1	1	1
01	0.53	0.58	0.97	0.93	0	0	1	1
10	0.23	0.84	0.62	0.64	1	1	0	1
11	0.33	0.83	0.27	0.95	1	1	1	0

In this case, the subfunctions are the clauses. Let f be a MAXkSAT expression of c clauses with variable selection masks $m_1, m_2, \ldots, m_c$ and negation masks $n_1, n_2, \ldots, n_c$, then

$$w_j^f = \begin{cases} c(1 - \frac{1}{2^k}) & \text{if } j = 0 \\ \frac{1}{2^k} \sum_{i\,:\,j \subseteq m_i} \psi_j(unpack(n_i, m_i)) & \text{if } j > 0 \end{cases} \tag{6.5}$$

where w_j is the j^{th} Walsh function. Notice that all of the Walsh coefficients for the single clause are nonzero and all of them except w_0 are of *exactly the same magnitude* but vary in sign. For example: for $k = 3$ all $w_j = \pm\frac{1}{8}$ for all $j \neq 0$. Since MAXSAT is an embedded landscape, the Walsh coefficients for a MAXSAT problem is simply the sum of the Walsh coefficients of the clauses. For randomly generated MAXkSAT problems, clauses tend to have equal numbers of negated and nonnegated variables. Therefore, the value of n in Equation 6.5 tends to be random. This causes w_j to have random sign. As the number of clauses, P, increases there is more and more opportunity for a given Walsh coefficient to be the sum of larger sets of contributing Walsh coefficients from clauses (Rana et al., 1998). The more subfunctions contributing to a single Walsh coefficient, the more the potential

values of the Walsh coefficients take on a binomial distribution about zero. Statistically, the lower the order of the Walsh coefficients of the embedded landscape, the more the Walsh coefficients of the subfunctions contribute to each Walsh coefficient for the whole landscape.

Since each clause in a MAXSAT problem is mostly 1 and returns a 0 for only one value of its arguments, the subfunctions in MAXSAT are mostly flat. The plateau configuration of the clauses combined with the fact that MAXSAT has the structure of an embedded landscape means MAXSAT problems are nearly flat. Since each subfunction is itself a plateau with a single minimum, each embedded function looks like a large plateau of fitness 1 with a single hyperplane basin of 0. Therefore the embedded landscape is a series of overlapping plateaus with very little fitness contrast. Subdivision of the space by embedded functions occurs by splitting the space into two regions of relative size $1 : 2^k - 1$. This means the largest plateau tends to only shrink as $((2^k - 1)/2^k)^P$ leaving a single large plateau and many smaller ones until the overlapping between subfunction masks interferes with this simplified model.

6.6 An Analysis of NK-landscapes

In the case of NK-landscapes, the role of the embedded functions is played directly by the functions $\frac{1}{N} r_j$. Each embedded function produces random fitness values generated uniformly between $[0, 1)$. Since the size of the masks used by the functions r_j are $K + 1$, the maximum level of epistasis for an NK-landscape must be $K + 1$.

The Walsh coefficient w_0 is in a tight normal distribution about 0.5. Since for all $w_i : i > 0 : \psi_i(x)$ for fixed x has an equal number of $+1$'s and -1's, w_i is the difference between the average of the 2^{L-1} random function values with positive Walsh functions and the average of the 2^{L-1} random function values with negative Walsh functions. Therefore, each average is approximately 0.5 with a normal distribution. The difference of the two distributions yields Walsh coefficients $w_i : i > 0$ that are all random and very near zero.

The random character of the embedded functions used in NK-landscapes causes each hyperplane to be assigned a different value in the implicit partitioning. When $P = 1$, there are 2^{K+1} hyperplanes each forming a plateau. As each embedded function is added,

the plateaus tend to be redivided into 2^{K-1} more hyperplanes, and the plateaus shrink exponentially in size by 2^{K-1}. In general, by the time $P = N/(2^{K-1})$, there are no plateaus left.

6.7 The NP-completeness of Embedded Landscapes

Although embedded landscapes can be quite limited in complexity, that doesn't make them easy. A decision language, L, is said to be **NP-hard** if

$$L' <_p L \text{ for every } L' \in \text{NP}$$

where $L' <_p L$ states that language L' *reduces to* L in polynomial time. A language L' reduces to L if every instance of a problem in L' can be re-expressed as an instance of a problem in L (Cormen et al., 1990). Let L_e be the language corresponding to the set of all embedded landscapes. It is well known (Papadimitriou 1994) that SAT and 3SAT are NP-hard and that

$$\text{SAT} <_p \text{3-SAT} <_p \text{MAX3SAT}.$$

Since, by definition, the language L_e contains every instance of the language MAX3SAT, it trivially follows that

$$\text{SAT} <_p \text{3-SAT} <_p \text{MAX3SAT} <_p L_e$$

and thus that L_e is NP-hard.

To be **NP-complete**, a language L must both be

1. NP-hard
2. $L \in NP$.

Is it true that $L_e \in$ NP? In general, the answer is no. For example, assume that the subfunction g_i takes k bits as input and that k is allowed to vary with N. If, for example,

$k = N/2$ then the subfunctions g_i are exponentially large with respect to N. Therefore, an arbitrary random function g_i may not have a compact representation: thus, it can only be described using exponential time/space with respect to N. In this case, the evaluation function has exponential cost. Thus, even a nondeterministic Turing machine cannot find or verify a solution in polynomial time. Therefore, in the general case, embedded landscapes are NP-hard, but not NP-complete. In this sense, embedded landscapes are more difficult than NP-complete problems.

The same argument holds for NK landscapes. If an NK landscape allows K to vary with N, then NK landscapes are not in the complexity class NP. Yet Weinberger (1996) proves that NK landscapes are NP-complete for $K \geq 3$. How is this possible? What Weinberger actually shows is that for any specific fixed K, NK landscapes are NP-complete. If K is fixed and not allowed to vary with N, then the size of each subfunctions g_i is of size 2^{K+1}. This may be a large or small constant, but it is a constant. The problem description for NK landscapes then has complexity $\mathcal{O}(N)$.

The same restriction can be placed on embedded landscapes. In practice, the only embedded landscapes which can be generated in polynomial time are such that

1. k is bounded by some constant
2. the *number* of subfunctions, P, is polynomial in N.

Note that when k is fixed, the number of possible subfunctions is automatically polynomial with respect to N. If every subfunction g_i uses exactly k bits, then every such function can be constructed using at most $P = \binom{N}{k}$ possible subfunctions.

Thus, when k is fixed, embedded landscapes have a polynomial time description. We will refer to the language over the subset of embedded landscapes with polynomial time descriptions as L_{pe}. An optimal solution to a problem in L_{pe} can be found or verified in polynomial time by a nondeterministic Turing machine. The verification is the same as the verification for *Traveling Salesperson Problems* (Cormen 1990): we must be told in advance the evaluation of the optimal solution. We then verify the solution we are given has that particular evaluation–which we can do since the evaluation function is polynomial. Therefore, $L_{pe} \in$ NP and it follows that:

1. L_e is NP-hard but not NP-complete since $MAX3SAT <_p L_e$ and $L_e \notin$ NP.
2. L_{pe} is NP-hard and NP-complete since $MAX3SAT <_p L_{pe}$ and $L_{pe} \in$ NP.

We can only be certain that an exact polynomial time Walsh analysis exists if those problems are taken from the language L_{pe}. Obviously, languages in L_e that do not allow a polynomial time description do not allow a polynomial time Walsh analysis.

Note that it is possible for all NP-complete problems to be expressed as MAXSAT problems. In particular, I showed, one can do a polynomial time Walsh analysis on MAX3SAT in time proportional to the problem description. This means once can have all of the Walsh coefficients in polynomial time relative to the number of bits in the domain. This means that we can also exactly compute all hyperplane averages up to any fixed order denoted by q. Furthermore, we know we can compute summary statistics for the function in polynomial time. Yet, with all of this information MAX3SAT is NP-complete. This means, if the complexity classes $P \neq$ NP, then having the Walsh coefficients, hyperplane averages, and summary statistics is not sufficient to guarantee finding the global optimum in polynomial time.

6.8 Summary

In this chapter I developed a very general form of function called an embedded landscape. It is composed of a sum of smaller embedded subfunctions. For subfunctions of bounded dimension the embedded landscape was shown to have limited epistasis and all its Walsh coefficients could be computed in polynomial time. I discussed other features of the structure of embedded landscapes including plateaus, various summary statistics, and a conjecture relating embedded landscapes to the Central Limit Theorem. I showed that both MAXSAT and NK-landscapes are forms of embedded landscapes and was able to carry over knowledge from the general embedded landscape to these specific cases.

The final section of this chapter revealed that by using MAXSAT as an example problem with low epistasis, it can be shown that problems with low epistasis could be NP-complete. Finally, I showed that, provided $P \neq$ NP, the knowledge gained by having the Walsh coefficients is insufficient to find the global optimum in polynomial time relative to the

number of bits in the domain.

Chapter 7

Conclusion

This dissertation has focused on computing the epistasis of problems via Walsh analysis and relating that to problem difficulty. The development of Walsh analysis has been with sufficient detail that this document can be used as a reference for Walsh analysis. Many new concepts have been introduced including: hyperplane numbering, *pack* and *unpack* functions, and function embedding. Various measures for epistasis are defined based on Walsh coefficients such as: Walsh sums, function order, and Walsh counts.

Unfortunately computing the Walsh coefficients for an arbitrary function takes exponential time even if the best known techniques are used. The result is that you can only have complete epistatic information if the problem is small or are by nature of limited epistasis. Fortunately, I proved that logical expressions and polynomials in combination with various encoding schemes can have such limited epistasis. I have also shown that a very general class of problems called embedded landscapes also has limited epistasis. In fact, embedded landscapes with subfunctions of with a fixed maximum number of bits in their domain can have all of their Walsh coefficients computed in polynomial time. I show that both MAXSAT and NK-landscapes are examples of embedded landscapes. Therefore, MAX3SAT can have all of its Walsh coefficients computed in polynomial time. I also show that MAX3SAT has an Ω of 3. But it is well known that MAX3SAT is NP-complete. Hence there exist problems that have low epistasis and all the Walsh coefficients are known, that are also NP-complete. I attribute this difficulty in part to MAX3SAT being a constraint satisfaction problem. Epistasis does not directly reflect this nature of problems.

7.1 Future Work

Even though epistasis can be a critical factor in making problems hard. it is not the only factor. For a clue of where to look next. we can turn to MAXSAT. the problem with low epistasis that was NP-complete. Apparently. the constraint satisfaction aspects of MAXSAT contributed heavily to the difficulty of that problem. A new question is whether the constraint satisfaction found in MAXSAT can be modeled similarly to epistasis giving a general theory of difficulty encompassing both. For MAXSAT. constraint satisfaction occurs as conflicts between groups of bits that epistatically interact. Consider the simple example of the MAXSAT problem where variables values are stored as bits in a string and the fitness function is $(a \vee x \vee y) + (\overline{x} \vee \overline{y} \vee b)$. The variables $a.x.y$ and $b.x.y$ epistatically interact. The variables a and b are also bound together but. not in the classic sense of epistasis. Examination of these constraint satisfaction cases show that conflict can be thought of as a type of loose epistasis. I call this loose epistasis **epistatic conflict**. One of my research goals will be to try to unify epistasis and the conflict found in constraint satisfaction using my previous work in epistasis as a starting point.

Another feature mentioned in the literature that makes problems difficult for genetic algorithms is deception (Goldberg. 1989b: Whitley. 1991: Deb and Goldberg. 1992). This occurs when a large basin of attraction for a local optima hides a much smaller basin of attraction for the global optima. Preliminary results show that deception can be modeled as simple competition between overlapping nondeceptive functions– one for each basin. This begs the question. can deception. as well. be modeled by the same super-epistatic model that models epistasis and epistatic conflict? How much of what we see as problem difficulty might be modeled under a single **super-epistatic model**?

A much studied feature of problem difficulty for kSAT problems is the phase transition between satisfiable and unsatisfiable problems (Kirkpatrick and Selman. 994). For 3SAT problems this occurs for problems with a clause to number of variable ratio of about 4.3. At the midpoint of this transition comes the point when the problems are by far more difficult for algorithms based on the Davis-Putnam algorithm (Davis and Putnam. 1960). This phase transition behavior needs to be verified for evolutionary algorithms. If these

problems are in fact getting difficult due to some extended measure of epistasis. then the super-epistatic model could give us new insights on phase transitions in problem difficulty.

One of the interesting features of embedded landscapes is the small number of bits that interact nonlinearly. Overlapping clusters form epistatic conflict. This basic structure is the same as the structure called the "small world" phenomenon that has recently been under study by Watts and Strogatz (Watts and Strogatz. 1998) of Cornell. but cast in the form of an optimization problem. I look forward to the opportunity to leverage their work to study epistatic conflict in optimization problems.

Since no estimation of problem difficulty can be made independently of the algorithm to be applied (Wolpert and Macready. 1997). I have created a preliminary algorithm test suite. In what sense is this test suite adequate? Does it need to be enlarged? Does it provide good coverage? Can it act as a preliminary means for categorizing evolutionary algorithms? The test suite should be expanded to contain problems that are derived from real world problems and yet simple enough in structure that there might be hope of associating features of the problem with difficulty.

In working on this dissertation. I developed software that performs various Walsh analysis functions. I used this software to empirically explore the relationship between epistasis and evolutionary algorithm performance. I also used this software to verify the identities resulting from most of the proofs. I have prepared my analysis software to maintain a database of example problems. the analysis of each. and performance marks for algorithms with various properties. This way extensive time consuming analysis of each problem need not be repeated and only new features and measures need to be cataloged.

The constructive use of neighborhood is not well understood. For example a plateau is only a threat to performance if the algorithm treats the domain of the plateau as a tightly connected region of space from which it is difficult to escape.

My work has been mostly in deducing dynamic behavior from static measures. A comparison of this work with more dynamic or process oriented measures such as approximate entropy (Hogg. 1998) and autocorrelation length should be performed.

How does chromosome length effect my results? What is the smallest problem that gen-

eralizes well to most large problems? How can this vague concept be made more concrete?

Can long tail performance prediction (Gomes et al., 1998) be used to improve the performance of evolutionary algorithms? What structural features of algorithm and problem exacerbate the long tail phenomenon?

But this is all work that must stand on a solid foundation. The most urgent of future work must be to firm up the work in this dissertation by application and experimentation: to support theory with experiment: to learn the extent of the application of techniques such as argument centering.

Finally, we now know that certain patterns of epistatic structure can cause a problem to be difficult for the kinds of stochastic search algorithms we have been discussing. But we are a long way from saying definitively what patterns cause what difficulties for what algorithms. This question has yet to be answered. It is not that I have failed to solve any large and important problem it is more that: in looking for the Holy Grail I have invented the Dixie cup. Not as glorious, but an elegant and very useful invention.

REFERENCES

Altenberg. L. (1994). Evolving better representations through selective genome growth. In *Proceedings of the IEEE World Congress on Computational Intelligence.* pages 182-187.

Altenberg. L. (1996). *NK Fitness Landscapes.* chapter B2.7.2. Oxford University Press. Oxford. England.

Bachman. G. (1964). *Elements of Abstract Harmonic Analysis.* Academic Press. Inc.. San Diego. CA.

Bäck. T.. Hoffmeister. F.. and Schwefel. H. (1991). A survey of evolution strategies. In Belew. R. K. and Booker. L. B.. editors. *Proceedings of the Fourth International Conference on Genetic Algorithms.* pages 2-9. Palo Alto. CA. Morgan Kaufmann Publishers. Inc.

Bagnall. T.. McKeown. G. P.. and Rayward-Smith. V. J. (1997). The cryptanalysis of a three rotor machine using a genetic algorithm. In Bäck. T.. editor. *Proceedings of the Seventh International Conference on Genetic Algorithms.* pages 712-718. Palo Alto. CA. Morgan Kaufmann Publishers. Inc.

Bethke. A. D. (1981). *Genetic Algorithms as Function Optimizers.* PhD thesis. University of Michigan. Department of Computer and Communication Sciences. Ann Arbor. Michigan.

Brittain. D.. Williams. J. S.. and McMahon. C. (1997). A genetic algorithm approach to planning the telecommunications access network. In Bäck. T.. editor. *Proceedings of the Seventh International Conference on Genetic Algorithms.* pages 623-628. Palo Alto. CA. Morgan Kaufmann Publishers. Inc.

Cormen. T. H.. Leiserson. C. E.. and Rivest. R. L. (1990). *Introduction to Algorithms.* McGraw-Hill. New York.

Davidor. Y. (1991). Epistasis variance: a viewpoint on ga-hardness. In Rawlins. G.. editor. *Foundations of Genetic Algorithms.* pages 93-108. Palo Alto. CA. Morgan Kaufmann Publishers. Inc.

Davis. L. (1991). *Handbook of Genetic Algorithms.* Van Nostrand Reinhold. New York.

Davis. M. and Putnam. H. (1960). A computing procedure for quantification theory. *Journal of the ACM.* 7:201-215.

Deb. K. and Goldberg. D. E. (1992). Analyzing deception in trap functions. In *foga2.* pages 93-108.

Deb. K. and Saxena. V. (1997). Car suspension design for comfort using a genetic algorithm. In Bäck. T.. editor. *Proceedings of the Seventh International Conference on Genetic Algorithms*. pages 553–560. Palo Alto. CA. Morgan Kaufmann Publishers. Inc.

Eiben. A. E. and van der Hauw. J. K. (1997). Solving 3-sat by gas adapting constraint weights. In *Proceedings of the 4th IEEE Conference on Evolutionary Computation*. pages 81–86. IEEE Service Center.

Eshelman. L. (1991). The chc adaptive search algorithm. how to have safe search when engaging in nontraditional genetic recombination. In Rawlins. G.. editor. *Foundations of Genetic Algorithms*. pages 265–283. Morgan Kaufmann Publishers. Inc.

Eshelman. L. and Schaffer. J. D. (1991). Preventing premature convergence in genetic algorithms by preventing incest. In Belew. R. K. and Booker. L. B.. editors. *Proceedings of the Fourth International Conference on Genetic Algorithms*.

Forrest. S. and Mitchell. M. (1992). Relative building-block fitness and the building-block hypothesis. In *Foundations of Genetic Algorithms - 2*. pages 109–126.

Goldberg. D. (1989a). Genetic algorithms and walsh functions: Part i. a gentle introduction. *Complex Systems*. 3:129–152.

Goldberg. D. (1989b). Genetic algorithms and walsh functions: Part ii. deception and its analysis. *Complex Systems*. 3:153–171.

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

Goldberg. D.. Korb. B.. and Deb. K. (1989). Messy genetic algorithms: Motivation. analysis. and first results. *Complex Systems*. 4:415–444.

Goldberg. D. E. and Rudnick. M. W. (1991). Genetic algorithms and the variance of fitness. *Complex Systems*. 5(3):265–278.

Gomes. C. P.. Selman. B.. and Kautz. H. (1998). Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. pages 431–437. Menlo Park. CA. AAAI Press.

Graham. R. L.. Knuth. D. E.. and Patashnik. O. (1989). *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Publishing. Co.. Reading. MA.

Heckendorn. R. B.. Rana. S.. and Whitley. D. (1999). Test function generators as embedded landscapes. In Bäck. T. and Banzhaf. W.. editors. *FOGA - 5*. Palo Alto. CA. Morgan Kaufmann Publishers. Inc.

Heckendorn. R. B. and Whitley. D. (1997). A walsh analysis of nk-landscapes. In Bäck. T.. editor. *Proceedings of the Seventh International Conference on Genetic Algorithms*. Palo Alto. CA. Morgan Kaufmann Publishers. Inc.

Heckendorn. R. B.. Whitley. L. D.. and Rana. S. (1997). Nonlinearity. hyperplane ranking and the simple genetic algorithm. In Belew. R. K. and Vose. M.. editors. *Foundations of Genetic Algorithms - 4*. Palo Alto. CA. Morgan Kaufmann Publishers. Inc.

Hogg. T. (1998). Which search problems are random? In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. pages 438–443. Menlo Park. CA. AAAI Press.

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

Hohn. F. E. (1969). *Applied Boolean Algebra: An Elementary Introduction.* MacMillan Company. Toranto. Canada.

Holland. J. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press.

Jones. T. (1995). *Evolutionary Algorithms. Fitness Landscapes and Search.* PhD thesis. University of New Mexico. Albuquerque. New Mexico.

Jong. K. A. D.. Potter. M. A.. and Spears. W. M. (1997). Using problem generators to expore the effects of epistasis. In Bäck. T.. editor. *Proceedings of the Seventh International Conference on Genetic Algorithms.* pages 338–339. Palo Alto. CA. Morgan Kaufmann Publishers. Inc.

Jong. K. A. D. and Spears. W. M. (1989). Using genetic algorithms to solve np-complete problems. In Schaffer. J. D.. editor. *Proceedings of the Third International Conference on Genetic Algorithms.* pages 124–132. Palo Alto. CA. Morgan Kaufmann Publishers. Inc.

Kauffman. S. (1989). Adaptation on rugged fitness landscapes. In Stein. D.. editor. *Lectures in the Science of Complexity.* pages 527–618. Addison-Wesley Publishing. Co.. Reading. MA.

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

Kauffman. S. A. (1995). *At Home in the Universe.* Oxford University Press. Oxford. England.

Kirkpatrick. S.. Jr.. C. D. G.. and Vecchi. M. P. (1983). Optimization by simulated annealing. *Science.* 220(4598):671–680.

Kirkpatrick. S. and Selman. B. (994). Critical behavior in the satisfiability of random boolean expressions. *Science.* 264:1297–1301.

Koza. J. R. (1992). *Genetic Programming : On the Programming of Computers by Means of Natural Selection.* MIT Press. Cambridge. MA.

Lechner. R. J. (1971). Harmonic analysis of switching functions. In Mukhopadhyay. A.. editor. *Recent Developments in Switching Theory.* pages 122–225. Academic Press. Inc.. San Diego. CA.

Manderick. B.. de Weger. M.. and Spiessens. P. (1991). The genetic algorithm and the structure of the fitness landscape. In Belew. R. K. and Booker. L. B.. editors. *Proceedings of the Fourth International Conference on Genetic Algorithms.* pages 143–150. Morgan Kaufmann Publishers. Inc.

Manela. M. and Campbell. J. A. (1992). Harmonic analysis. epistasis and genetic algorithms. In *Parallel Problem Solving from Nature. 2.* pages 57–64. Amsterdam. Holland. Elsevier Science Publishers.

Maresky. J.. Davidor. Y.. Gitler. D.. Aharoni. G.. and Barak. A. (1995). Selectively destructive re-start. In *Proceedings of the Sixth International Conference on Genetic Algorithms.* pages 144–150. Palo Alto. CA. Morgan Kaufmann Publishers. Inc.

Mathias. K. E. and Whitley. L. D. (1994). Transforming the search space with gray coding. *IEEE Transactions on Evolutionary Computation.* 1:513–518.

Mendelson. E. (1970). *Boolean Algebra and Switching Theory.* Schaum's Outline Series. McGraw-Hill. New York.

Mendenhall. W. (1967). *Introduction to Probability and Statistics.* Wadsworth Publishing Company. Inc.. Belmont. CA. second edition.

Meysenburg. M. and Foster. J. (1997). The quality of pseudo-random number generators and simple genetic algorithm performance. In Bäck. T.. editor. *Proceedings of the Seventh International Conference on Genetic Algorithms.* pages 276–282. Palo Alto. CA. Morgan Kaufmann Publishers. Inc.

Mitchell. D.. Selman. B.. and Levesque. H. (1992). Hard and easy distribution of sat problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence.* San Jose. CA.

Mitchell. M. (1996). *An Introduction to Genetic Algorithms.* MIT Press. Cambridge. MA.

Mühlenbein. H.. Schomisch. M.. and Born. J. (1991). The parallel genetic algorithm as a function optimizer. In Belew. R. K. and Booker. L. B.. editors. *Proceedings of the Fourth International Conference on Genetic Algorithms.* pages 271–278. Morgan Kaufmann Publishers. Inc.

Mülenbein. H. and Schlierkamp-Voosen. D. (1993). Predictive models for the breeder genetic algorithm. *Evolutionary Computation.* 1(1):25–49.

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

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

Petit. C. W. (1998). Touched by nature. *U.S. News & World Report.*

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. Nominated for the AAAI98 Outstanding Paper Award.

Rana. S. and Whitley. L. D. (1997). Bit representations with a twist. In Bäck. T.. editor. *Proceedings of the Seventh International Conference on Genetic Algorithms.* pages 188–195. Palo Alto. CA. Morgan Kaufmann Publishers. Inc.

Rao. S. S. (1998). Evolution at warp speed. *Forbes.*

Reeves. C. and Wright. C. (1995a). Epistasis in genetic algorithms: An experimental design perspective. In Eschelman. L. J.. editor. *Proceedings of the Sixth International Conference on Genetic Algorithms.* pages 217–224. Morgan Kaufmann Publishers. Inc.

Reeves. C. and Wright. C. (1995b). An experimental design perspective on genetic algorithms. In Whitley. D. and Vose. M.. editors. *Foundations of Genetic Algorithms - 3.* pages 7–22. Morgan Kaufmann Publishers. Inc.

Schwefel. H.-P. and Bäck. T. (1998). Artificial evolution: How and why? In Quagliarella. D., Periaux. J.. Poloni. C.. and Winter. G.. editors. *Genetic Algorithms and Evolution*

Strategies in Engineering and Computer Science. pages 1-19. John Wiley. Chichester. England.

Smith. R. E. and Smith. J. E. (1999). An exmaination of tunable. random search landscapes. In Bäck. T. and Banzhaf. W.. editors. *FOGA - 5*. pages 165-181. Palo Alto. CA. Morgan Kaufmann Publishers. Inc.

Spiegel. M. R. (1961). *Statistics*. Schaum's Outline Series. McGraw-Hill. New York.

Todd. D. S. and Sen. P. (1997). A multiple criteria genetic algorithm for containership loading. In Bäck. T.. editor. *Proceedings of the Seventh International Conference on Genetic Algorithms*. pages 674-681. Palo Alto. CA. Morgan Kaufmann Publishers. Inc.

Tsutsui. S.. Ghosh. A.. Corne. D.. and Fujimoto. Y. (1997). A real coded genetic algorithm with an explorer and an exloiter populations. In *Proceedings of the Seventh International Conference on Genetic Algorithms*. pages 238-245. Palo Alto. CA. Morgan Kaufmann Publishers. Inc.

Venturini. G.. Slimane. M.. Morin. F.. and de Beaufille. J.-P. A. (1997). On using interactive genetic algorithms for knowledge discovery in databases. In Bäck. T.. editor. *Proceedings of the Seventh International Conference on Genetic Algorithms*. pages 696-703. Palo Alto. CA. Morgan Kaufmann Publishers. Inc.

Voget. S. (1995). Epistasis in periodic programs. In Eschelman. L. J.. editor. *Proceedings of the Sixth International Conference on Genetic Algorithms*. pages 225-230. Morgan Kaufmann Publishers. Inc.

Vose. M. and Liepins. G. (1991). Punctuated equilibria in genetic search. *Complex Systems*. 5:31-44.

Watts. D. J. and Strogatz. S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*. 393:440-442.

Weaver. H. J. (1989). *Theory of Discrete and Continuous Fourier Analysis*. John Wiley. Chichester. England.

Weinberger. E. (1990). Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*. 63:325-226.

Whitley. D. (May 26. 1999). Personal communication.

Whitley. D.. Mathias. K.. and Pyeatt. L. (1995a). Hyperplane ranking in simple genetic algorithms. In Eschelman. L. J.. editor. *Proceedings of the Sixth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers. Inc.

Whitley. D.. Mathias. K.. Rana. S.. and Dzubera. J. (1995b). Building better test functions. In Eshelman. L.. editor. *Proceedings of the Sixth International Conference on Genetic Algorithms*. pages 239-246. Palo Alto. CA. Morgan Kaufmann Publishers. Inc.

Whitley. L. D. (1989). The genitor algorithm and selective pressure: Why rank based allocation of reproductive trials is best. In Schaffer. J. D.. editor. *Proceedings of the Third International Conference on Genetic Algorithms*. pages 116-121. Morgan Kaufmann Publishers. Inc.

Whitley. L. D. (1991). Fundamental principles of deception in genetic search. In Rawlins. G.. editor. *Foundations of Genetic Algorithms*. pages 221-241. Morgan Kaufmann Publishers. Inc.

Whitley. L. D. (1993). An executable model of the simple genetic algorithm. In Whitley. L. D.. editor. *Foundations of Genetic Algorithms - 2*. pages 45–62. Morgan Kaufmann Publishers. Inc.

Wolpert. D. H. and Macready. W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*. 1(1):67–82.

INDEX