Linear Separability Of The Vertices Of An n-Dimensional Hypercube

Nicolle Gruzling

B.Sc., University of Northern British Columbia, 2001

Thesis Submitted In Partial Fulfillment Of

The Requirements For The Degree Of

Master of Science

in

Mathematical, Computer, and Physical Sciences

(Computer Science)

The University of Northern British Columbia

December 2006

Your file Votre référence
ISBN: 978-0-494-28428-5
Our file Notre référence
ISBN: 978-0-494-28428-5

Abstract

Boolean Algebra is the basis for many fields of study, including a Artificial Intelligence, Game Theory, Coding Theory, and Distributed Algorithms. Pioneering work by Muroga and Winder set limits to the number of n variable threshold functions. This thesis expands on their work by calculating the number of 9 variable threshold functions, found to be 144,130,531,453,121,108. New methods of finding separating weight vectors for threshold functions are examined. A discrepancy between Winder's defined Chow Parameters and the Chow Parameters calculated is corrected within the thesis, resulting in better weight vector performance in separating threshold functions. Of the methods examined Chow Parameters are found to be the most effective weight vectors in separating threshold functions.

Contents

Abstract ii

List of Tables vi

List of Figures viii

1 Literature Review 1
- 1.1 Introduction . . . 1
- 1.2 Threshold Logic . . . 6
  - 1.2.1 Boolean Functions . . . 6
  - 1.2.2 Threshold Functions . . . 7
  - 1.2.3 Monotonicity . . . 11
  - 1.2.4 Unateness . . . 11
  - 1.2.5 Asummability . . . 12
- 1.3 Chow Parameters . . . 13
- 1.4 Equivalence Classes of Boolean Functions . . . 18
  - 1.4.1 Number of Self-dual Boolean Functions . . . 26
  - 1.4.2 Generation of Canonical Sets . . . 27
- 1.5 Classifying Threshold Functions . . . 27
  - 1.5.1 Classification Using Chow Parameters . . . 30
- 1.6 Number of Threshold Functions . . . 32
  - 1.6.1 Threshold numbers . . . 35
- 1.7 Applications of Threshold Functions . . . 35
  - 1.7.1 Artificial Neural Networks . . . 36
  - 1.7.2 Simple Games . . . 38
  - 1.7.3 Convex Polytopes . . . 40
  - 1.7.4 Complexity of Boolean Functions . . . 40
  - 1.7.5 A Few More Applications . . . 41

2 Canonical Set Generation **42**
2.1 Representation Of A Set 42
2.2 Method of Muroga, Toda, and Kondo 43
2.3 Winder's Method 46

3 Classifying Threshold Functions **52**
3.1 Number of Threshold Functions 53
3.1.1 Simplex Method and Separability 53
3.1.2 Number of Threshold Functions 56
3.2 Chow Parameters and Separability 57
3.2.1 Derivation of Chow Parameters 58
3.2.2 Adaptations of Chow Parameters 62
3.3 Border Sums and Separability 66

4 Conclusion **68**
4.1 Generation of Boolean Functions 68
4.2 Number of Threshold Functions 69
4.3 Chow Parameters and Linear Separability 70
4.4 Border and Boundary Points 70

Bibliography **71**

List of Tables

1.1 Functions from $\{0,1\} \mapsto \{0,1\}$. 1

1.2 Functions from $\{0,1\}^2 \mapsto \{0,1\}$. 2

1.3 Number of NP-equivalent types of Boolean functions of n variables [Slepian 1953]. 19

1.4 Self-dualizations of the $n = 1$ variable Boolean functions. 21

1.5 Number of P, NP, NPN, and SD equivalence classes containing Boolean functions of n variables [Muroga 1971]. 24

1.6 Determining separability using 7 different methods [Winder 1971]. Column 2 lists the number of functions that were correctly classified as threshold functions, and Column 3 lists the average number of elements that differ between the actual function and the function calculated by the weight vector. 31

1.7 Actual Numbers of Threshold Functions [Muroga 1971] 32

2.1 Number of Boolean functions that need to be tested for linear separability using the method given by Muroga et al [1962]. 45

2.2 Number of hypercomplete Boolean functions generated by Winder's algorithm compared to the number of complete Boolean functions. . . 51

3.1 Number of hypercomplete threshold functions of n variables, denoted by $HN(n)$. 56
3.2 Numbers of threshold functions of n variables. 57
3.3 Excerpt of the table given by Winder [1969]. 59
3.4 Number of threshold functions that are separated correctly when using Chow parameters as a weight vector. $N(n)$ is the number of hypercomplete threshold functions of n variables. 62
3.5 Number of threshold functions that are separated correctly when using functions of Chow parameters as a weight vector. $N(n)$ is the number of hypercomplete threshold functions of n variables. 65
3.6 Number of threshold functions that are separated correctly when using border sums to derive weight vectors. 67

List of Figures

1.1 Pictorial representations of $f_1(x, y) = xy$ and $f_3(x, y) = x$. 3

1.2 Pictorial representations of $f_6(x, y) = x \oplus y$ and $f_9(x, y) = x \odot y$. . . . 4

1.3 The threshold function $f(x, y, z) = 0$. 4

1.4 Threshold functions of 3 variables. 5

1.5 Some 3 variable functions that are not threshold functions. 5

1.6 Equivalent sets $F_1, F_2, F_3 \subseteq \mathbb{K}_3$. 18

1.7 Boolean functions of $n = 1$ variable. 21

1.8 Boolean functions of $n = 1$ variable separated into P, NP, NPN, and SD equivalence classes. 22

1.9 Self-dualizations of the $n = 1$ variable Boolean functions. 22

1.10 The SD class containing f_0, f_1, f_2, f_3. 23

1.11 Logic gates evaluating f, f^d, and f^{sd} such that $f : \{-1, 1\}^n \mapsto \{-1, 1\}$. 25

1.12 A Multiple-Input Neuron: $a = f(w_1x_1 + w_2x_2 + \cdots + w_nx_n - b)$. . . . 37

2.1 $A = \{000, 001, 101\} \subset \mathbb{K}_3$ is written 00100011. 43

2.2 The lattice, L_2, defined by Winder [1965]. 47

Chapter 1

Literature Review

1.1 Introduction

A Boolean function of n variables is a function with inputs and output from any 2-valued set such as $\{0,1\}$, $\{-1,1\}$ or {TRUE,FALSE}. We will be using $\{0,1\}$ unless otherwise stated. A Boolean function is defined with a truth table or with Boolean logic operators such as AND, OR, and NOT which are, in fact, Boolean functions of 1 or 2 variables. Table 1.1 lists all 4 of the 1-variable Boolean functions; $f_0 = 0$, $f_3 = 1$ are constant functions, $f_1 = x$ is the identity function and $f_2 = \overline{x}$ is the complementation function.

x	f_0	f_1	f_2	f_3
0	0	0	1	1
1	0	1	0	1

Table 1.1: Functions from $\{0,1\} \mapsto \{0,1\}$.

x	y	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Table 1.2: Functions from $\{0,1\}^2 \mapsto \{0,1\}$.

In general there are 2^{2^n} Boolean functions of n variabes, so for $n = 2$ there are 16 Boolean functions (all listed in Table 1.2). Some familiar 2-variable Boolean functions are the OR function[1] ($f_7(x,y) = x+y$), the AND function[2] ($f_1(x,y) = xy$), and the XOR function[3] ($f_6(x,y) = x \oplus y$).

Definition 1. The set $\{0,1\}^n$ is called a **hypercube of n dimensions**. For $\mathbf{w} \in \mathbb{R}^n$ and $t \in \mathbb{R}$ such that $\mathbf{w}$ is non-zero, the set $\{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{w} \cdot \mathbf{x} = t\}$ is called a **hyperplane** in $\mathbb{R}^n$. Every hyperplane **separates** the vertices of the hypercube, with the vertices $\mathbf{x} \in \{0,1\}^n$ such that $\mathbf{w} \cdot \mathbf{x} \geq t$ on one side of the hyperplane and the rest on the other.

We will be concerned with a class of Boolean functions called threshold functions. If we consider the possible inputs of a Boolean function to be the vertices of an n-dimensional hypercube we can represent the output by colouring a vertex black if the function output is 1 and white if the function output is 0. When it is possible to separate the black and white vertices with a hyperplane, the function is a threshold function.

For 2-variable Boolean functions the hypercube is a square and the separating

[1]This functions can also be written $x \vee y$ or $x|y$ (C++ syntax).
[2]Also known as $x \cdot y$, $x \wedge y$, $x\&y$ (C++).
[3]Written $x\hat{}y$ in C++.

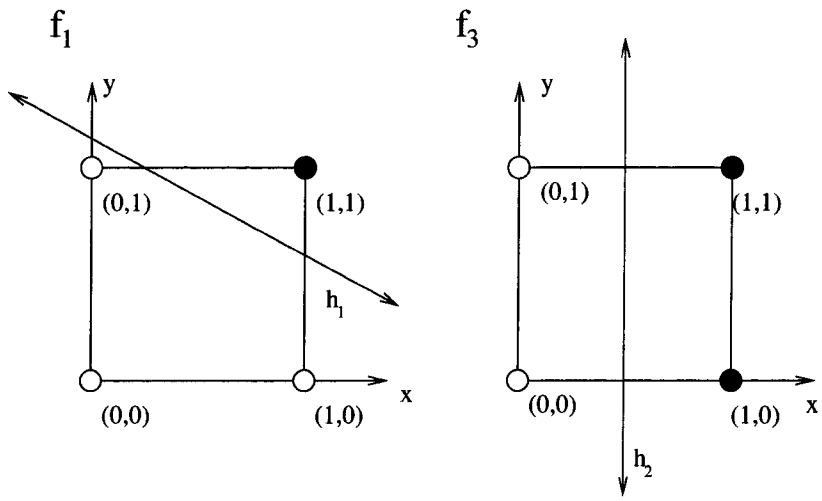

Figure 1.1: Pictorial representations of $f_1(x, y) = xy$ and $f_3(x, y) = x$.

hyperplane is a line. Figure 1.1 depicts 2-dimensional hypercubes embedded in $\mathbb{R}^2$ and the Boolean functions $f_1(x, y) = xy$ and $f_3(x, y) = x$ along with respective separating hyperplanes h_1 and h_2. There are only 2 Boolean functions of 2 variables that are not linearly separable, these are $f_6(x, y) = x \oplus y$ and $f_9(x, y) = \overline{x \oplus y} = x \odot y$. These are both shown in Figure 1.2 where it is easy to see that there is no line that can separate the black vertices from the white vertices.

It is much more difficult to classify a Boolean function as a threshold fuction when the function has more that 2 variables. Figure 1.3 illustrates $f(x, y, z) = 0$, this figure is included to show the labelling of 3 dimensions; other diagrams omit these labels. Figure 1.4 illustrates some examples of 3 variable threshold functions, while Figure 1.5 depicts some Boolean functions that are not threshold functions. It is the objective of this thesis to provide some insight into what characterizes a threshold function.

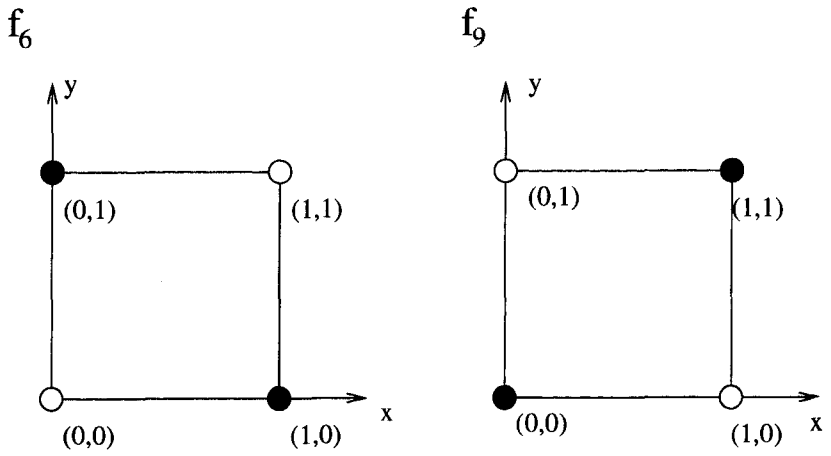

Figure 1.2: Pictorial representations of $f_6(x, y) = x \oplus y$ and $f_9(x, y) = x \odot y$.

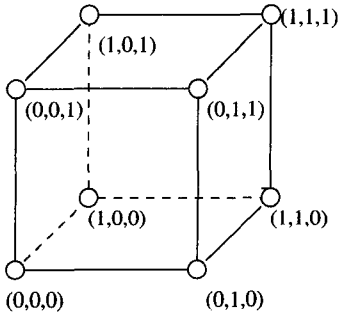

Figure 1.3: The threshold function $f(x, y, z) = 0$.

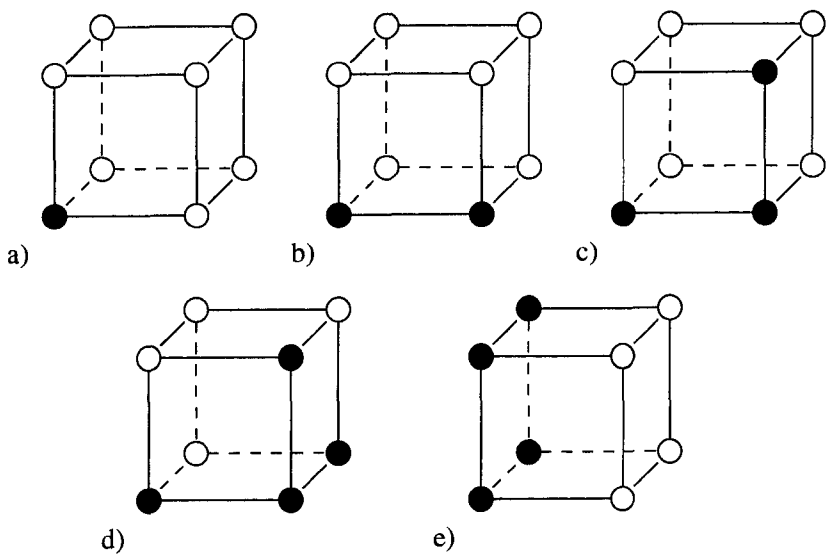

Figure 1.4: Threshold functions of 3 variables.

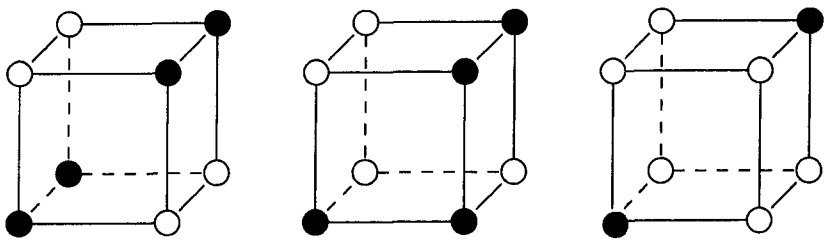

Figure 1.5: Some 3 variable functions that are not threshold functions.

1.2 Threshold Logic

1.2.1 Boolean Functions

Let $\mathbb{K}$ be the set $\{0,1\}$ and $\mathbb{K}_n$ the n-dimensional hypercube $\{0,1\}^n$. The elements or vectors of $\mathbb{K}_n$ are n-tuples denoted by $(x_1, x_2, \ldots, x_n)$ or $x_1x_2\cdots x_n$ with every $x_i \in \mathbb{K}$ and $i \in \{1, 2, \ldots, n\}$. Since $x_1x_2\cdots x_n$ is a binary sequence every element corresponds to an integer x with $0 \leq x \leq 2^n - 1$. Also $\mathbb{K}$ is a subset of $\mathbb{R}$ and the n-dimensional hypercube $\mathbb{K}_n$ is a subset of $\mathbb{R}^n$.

Definition 2. A **Boolean function** or **switching function of n variables** is a function, $f\colon \mathbb{K}_n \rightarrow \mathbb{K}$, from the n-dimensional hypercube $\mathbb{K}_n$ into $\mathbb{K}$. For the Boolean function, f, any vector $\mathbf{x} \in f^{-1}(1)$ is called a **true vector** of f and any vector $\mathbf{x} \in f^{-1}(0)$ is called a **false vector** of f.

Let Φ be the set of all switching functions of n variables. Then Φ, along with the unary operator complementation and binary operators conjunction and disjunction, is a Boolean algebra. Similiarly, the set Ψ of all possible subsets of $\mathbb{K}_n$, along with set union, intersection and complementation with respect to $\mathbb{K}_n$, is also a Boolean algebra. It has been shown by Hu [1965] that Φ and Ψ are isomorphic and every switching function f can be converted into a subset $F \subseteq \mathbb{K}_n$ (and vice versa) using the following transformations:

$$\rho\colon \Phi \rightarrow \Psi \quad \rho(f) = f^{-1}(1)$$

$$\theta\colon \Psi \to \Phi \quad \theta(F) = \chi_F \text{ where } \chi_F(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{x} \in F \\ 0, & \text{if } \mathbf{x} \notin F \end{cases}$$

We will be using subsets of $\mathbb{K}_n$ and Boolean functions of n-variables interchangeably.

1.2.2 Threshold Functions

Definition 3. A set $F \subseteq \mathbb{K}_n$ is **linearly separable** if there exists a non-zero $\mathbf{w} \in \mathbb{R}^n$ and $t \in \mathbb{R}$ such that

$$\mathbf{w} \cdot \mathbf{x} \geq t \text{ for all } \mathbf{x} \in F$$

and

$$\mathbf{w} \cdot \mathbf{x} < t \text{ for all } \mathbf{x} \in \bar{F}.$$

The corresponding Boolean function $f\colon \mathbb{K}_n \to \mathbb{K}$ is called a **threshold function** (or a **linearly separable function** or a **realizable function** or a **setting function**). The system $(\mathbf{w}, t)$ is called a **separating system** for F and f.

Definition 4. [Hu 1965] For a set $S \subseteq \mathbb{R}^n$, the **convex hull of** S is the smallest convex set of $\mathbb{R}^n$ which contains S.

Every threshold function has an infinite number of separating systems and it is well known that for every threshold function there exists a separating system whose weights and threshold are integers. In geometric terms a switching function f is linearly separable if and only if the convex hulls of $F = f^{-1}(1)$ and $\bar{F} = f^{-1}(0)$ are disjoint [Hu 1965] [Taylor and Zwicker 1999].

The following definitions will help in discussing the properties of Boolean functions.

Definition 5. The **dual** of a Boolean function f of n variables, denoted by f^d, is defined as

$$f^d(\mathbf{x}) = \bar{f}(\bar{\mathbf{x}})$$

where $\bar{f}$ is the complement of f and $\bar{\mathbf{x}} = (\bar{x_1}, \bar{x_2}, \ldots \bar{x_n})$. If $f = f^d$ then f is called a **self-dual function**. For the corresponding set $F = f^{-1}(1)$, the dual $F^d = \{\mathbf{x} \in F \mid \bar{\mathbf{x}} \in \bar{F}\}$.

Definition 6. For Boolean functions f and g (both of n variables), we say f **implies** g, denoted $f \leq g$ if and only if for every point $\mathbf{x} \in \mathbb{K}_n$ $f(\mathbf{x}) \leq g(\mathbf{x})$. In other words if $f(\mathbf{x}) = 1$ then $g(\mathbf{x}) = 1$ as well. In some literature $f \leq g$ is written $f \subseteq g$. Also f and g are considered **comparable** if $f \leq g$ or $g \leq f$. For the corresponding sets $F = f^{-1}(1)$ and $G = g^{-1}(1)$, $f \leq g$ if and only if $F \subseteq G$. Simliarly F and G are comparable when $F \subseteq G$ or $G \subseteq F$.

Definition 7. Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. Then $\mathbf{x}$ is said to **precede** $\mathbf{y}$, denoted $\mathbf{x} \preceq \mathbf{y}$, if $x_i \leq y_i$ for all $i \in \{1, 2, \ldots, n\}$. Also, $\mathbf{x}$ **strictly precedes** $\mathbf{y}$, denoted by $\mathbf{x} \prec \mathbf{y}$, if $\mathbf{x} \preceq \mathbf{y}$ and $x_i < y_i$ for some $j \in \{1, 2, \ldots, n\}$.

Definition 8. [Hu 1965] Let $\mathbb{K}_* = \{0, 1, *\}$ and $N = \{1, 2, \ldots, n\}$. For any function ϕ: $N \rightarrow \mathbb{K}_*$ a subset C of $\mathbb{K}_n$, called a **cube** in $\mathbb{K}_n$, is determined as follows. A point $\mathbf{x} = x_1 x_2 \cdots x_n \in \mathbb{K}_n$ is in $C \subseteq \mathbb{K}_n$ if and only if $x_i = \phi(i)$ for all $i \in N$ such that $\phi(i) \neq *$. The function ϕ is called the **coordinate function** of the cube C and cube $C \subseteq \mathbb{K}_n$ is denoted by $C = \phi(1)\phi(2)\cdots\phi(n)$. The **dimension** of a cube $C \subseteq \mathbb{K}_n$ is the cardinality of $\phi^{-1}(*) \subseteq N$, denoted by $dim(C) = |\phi^{-1}(*)|$. Cubes of dimension r are called r-**cubes**. The **codimension** of C, $cod(C) = n - dim(C)$.

The coordinates $i \in \{1, 2, \ldots, n\}$ such that $\phi(i) = *$ are called **free coordinates** and the other coordinates are called **fixed coordinates**.

There are 3^n cubes in the n-dimensional hypercube $\mathbb{K}_n$ and the number of r-cubes in $\mathbb{K}_n$ is $\frac{n!2^{n-r}}{r!(n-r)!}$[Hu 1965].

Definition 9. [Hu 1965] For r-cubes $C, D \subseteq \mathbb{K}_n$, C and D are **consecutive** if and only if they differ in only one coordinate. That is, there is a j such that $\phi_C(i) = \phi_D(i)$ for all $i \neq j$ and $\phi_C(j) = \overline{\phi_D(j)}$ with $\phi_C(j) \neq *$. Consecutive r-cubes C, D are called **opposite faces** of the $(r+1)$-cube $C \cup D$.

Definition 10. [Hu 1965] Let $C = \phi(1)\phi(2)\cdots\phi(n) \subseteq \mathbb{K}_n$ be an r-cube such that $r \in \{1, 2, \ldots, n\}$. A pair of points $\mathbf{x}, \mathbf{y} \in C$ are said to be a **diagonal of** C if and only if $x_i = \bar{y}_i$ for every $i \in \{1, 2, \ldots, n\}$ such that $\phi(i) = *$.

It is shown by Hu [1965] that for a linearly separable subset $F \subseteq \mathbb{K}_n$, if F contains a diagonal of an r-cube C in $\mathbb{K}_n$, then it contains both points of one diagonal and at least one point of every other diagonal of C and thus contains more than 2^{r-1} points.

Definition 11. [Hu 1965] Cubes $C, D \subseteq \mathbb{K}_n$, where $C = \phi(1)\phi(2)\cdots\phi(n)$ and $D = \psi(1)\psi(2)\cdots\psi(n)$, are **isomerous** if and only if $\phi^{-1}(*) = \psi^{-1}(*)$. C and D are **opposite** if and only if

$$\psi(i) = \begin{cases} \phi(i) & \text{if } \phi(i) = * \\ 1 - \phi(i) & \text{if } \phi(i) \neq *. \end{cases}$$

So opposite cubes are isomerous.

Definition 12. [Hu 1965] A cube $C = \phi(1)\phi(2)\cdots\phi(n) \subseteq \mathbb{K}_n$ such that $cod(C) = 1$ and with fixed coordinate $i \in \{1, 2, \ldots, n\}$ is called the **i-th upper face** of $\mathbb{K}_n$ if $\phi(i) = 1$ or the **i-th lower face** of $\mathbb{K}_n$ if $\phi(i) = 0$. The i-th upper and lower faces are consecutive and opposite.

Definition 13. [Hu 1965] Let $F \subseteq \mathbb{K}_n$ and $C = \phi(1)\phi(2)\cdots\phi(n) \subseteq \mathbb{K}_n$ be an m-cube. For $M = \{1, 2, \ldots m\}$ and $N = \{1, 2, \ldots, n\}$ let

$$\mu\colon \phi^{-1}(*) \to M$$

be the unique monotonic function from $\phi^{-1}(*) \subset N$ onto M. Now let $\iota\colon \mathbb{K}_m \to \mathbb{K}_n$ be defined such that for each vector $\mathbf{x} = x_1x_2\cdots x_m \in \mathbb{K}_m$, the coordinates of $\mathbf{y} = y_1y_2\cdots y_n = \iota(\mathbf{x}) \in \mathbb{K}_n$ are given by

$$y_i = \begin{cases} \phi(i) & \text{if } \phi(i) \neq * \\ x_{\mu(i)} & \text{if } \phi(i) = *. \end{cases}$$

Then $F_C \subseteq \mathbb{K}_m$ called the **restriction of F over C** is defined as

$$F_C = \{\iota^{-1}(\mathbf{x}) \mid \mathbf{x} \in F \wedge \mathbf{x} \in C\}$$

and for Boolean function f such that $f^{-1}(1) = F$ the **restriction of f over C** is $f_C = f \circ \iota\colon \mathbb{K}_m \to \mathbb{K}$. In particular if C is the i-th upper face of $\mathbb{K}_n$ then F_C or f_C is called the **i-th upper residue** of f and is denoted F^i or f^i, or if C is the i-th lower face of $\mathbb{K}_n$ then F_C or f_C is called the **i-th lower residue** of f and is denoted F_i or f_i.

1.2.3 Monotonicity

Definition 14. A Boolean function f of n variables is **k-monotonic** if and only if for any 2 isomerous cubes $C, D \subseteq \mathbb{K}_n$ with $cod(C) \leq k$, the restrictions f_C and f_D are comparable. If f is k-monotonic for all $k \in \{1, 2, \ldots, n\}$ then f is **completely monotonic**. The corresponding set $F = f^{-1}(1)$ is called k-monotonic if f is k-monotonic and completely monotonic if f is completely monotonic.

It was shown first by Paull and McCluskey [1960] that all threshold functions are completely monotonic. At the time it was also conjectured that all completely monotonic functions were threshold functions. The first counterexample was a 12-variable Boolean function provided by E.F. Moore in an unpublished memorandum. All completely monotonic Boolean functions of n variables with $n \leq 8$ are, in fact, threshold functions [Muroga 1971], but for $n = 9$ there are completely monotonic functions that are not threshold functions. Discovery of these examples is credited to Gabelman (see [Muroga 1971] or [Taylor and Zwicker 1999] for examples). It was shown by Winder [1961] that if a Boolean function f of n variables is k-monotonic for $k \geq \lfloor n/2 \rfloor$ then f is completely monotonic.

1.2.4 Unateness

Definition 15. A Boolean function f of n variables is called **positive**[4] if $f(\mathbf{x}) \geq f(\mathbf{y})$ whenever $\mathbf{x} \succeq \mathbf{y}$ for $\mathbf{x}, \mathbf{y} \in \mathbb{K}_n$.

Definition 16. [Bshouty 1995] [Boros et al, 2003] For a fixed vector $\mathbf{b} \in \mathbb{K}_n$ and $\mathbf{x}, \mathbf{y} \in \mathbb{K}_n$, define a partial order $\succeq_{\mathbf{b}}$ by $\mathbf{x} \succeq_{\mathbf{b}} \mathbf{y}$ if and only if $\mathbf{x} \oplus \mathbf{b} \succeq \mathbf{y} \oplus \mathbf{b}$, where

[4] A positive function is also called monotone in recent literature.

$\oplus$ is componentwise XOR. A Boolean function f is **b-monotone** if $f(\mathbf{x}) \geq f(\mathbf{y})$ whenever $\mathbf{x} \succeq_{\mathbf{b}} \mathbf{y}$. Notice that a function is positive when it is **0**-monotone.

Definition 17. A Boolean function f of n variables is said to be **unate** if there exists $\mathbf{b} \in \mathbb{K}_n$ such that f is **b**-monotone.

It was shown by Winder [1961] that a Boolean function of n variables is unate if and only if it is 1-monotonic. Since every threshold function is completely monotonic it follows that every threshold function is unate. When testing whether a given function f is a threshold function, a test for unateness is usually the first test performed. In Section 1.3 we will define Chow parameters. If a Boolean function is **b**-monotone then it is easy to determine **b** (which is not necessarily unique) from the Chow parameters. This simplifies the test for unateness.

1.2.5 Asummability

Definition 18. Let $F \subseteq \mathbb{K}_n$ and $\bar{F} = \mathbb{K}_n - F$. Then F (and the corresponding Boolean function f) is **k-summable** for $k \geq 2$ if and only if there exists j, $2 \leq j \leq k$ such that there are j (not necessarily distinct) points $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_j \in F$ and j (not necessarily distinct) points $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_j \in \overline{F}$ such that

$$\sum_{i=1}^{j} \mathbf{u}_i = \sum_{i=1}^{j} \mathbf{v}_i.$$

If F is not k-summable, then F is said to be **k-asummable**. F is said to be **summable** if and only if it is k-summable for some $k \geq 2$, otherwise F is **asummable**.

Theorem 1. *[Elgot 1961] A Boolean function f of n variables is a threshold function*

if and only if it is asummable.

A similiar result was given by Chow [1961a]. These conditions for separability follow from linear algebra considerations.

Theorem 2. *[Chow 1961a] For $F \subseteq \mathbb{K}_n$, where $\mathbf{v_1}, \ldots, \mathbf{v_m}$ are the true vectors and $\mathbf{v_{m+1}}, \ldots, \mathbf{v_{2^n}}$ are the false vectors, F is linearly separable if and only if for any non-negative numbers $c_i \geq 0$ where $1 \leq i \leq 2^n$ the relations*

$$\sum_{i=1}^{m} c_i = \sum_{i=m+1}^{2^n} c_i$$

and

$$\sum_{i=1}^{m} c_i \mathbf{v_i} = \sum_{i=m+1}^{2^n} c_i \mathbf{v_i}$$

imply $c_i = 0$ for all $i = 1, 2, \ldots, 2^n$.

The conditions in Theorems 1 and 2 both imply that the convex hulls of F and $\bar{F}$ are disjoint. In Section 1.2.3 monotonicity was defined. Theorem 3 relates monotonicity and asummability.

Theorem 3. *[Elgot 1961] A Boolean function f of n variables is completely monotonic if and only if it is 2-asummable.*

1.3 Chow Parameters

Chow [1961b] defined what are now referred to as Chow parameters. Similiar parameters had been defined in earlier papers. For instance, Golomb [1959] defined a

set of 2^n invariants where the first $n+1$ invariants [5] are were essentially Chow parameters, but since Chow provided the seminal theorems related to the parameters, it is always his name that is attached to these types of parameters.

Definition 19. Let f be a Boolean function of n variables and $F = f^{-1}(1)$. The **Chow parameters** for F (or f) are a pair $(m_F, \mathbf{a}_F)$ (or $(m_f, \mathbf{a}_f)$ or just $(m, \mathbf{a})$ if F is self-evident) with $m_F \in \mathbb{N}$ and $\mathbf{a}_F \in \mathbb{N}^n$ such that

$$m_F = |F|$$

and

$$\mathbf{a}_F = (a_{F1}, a_{F2}, \ldots, a_{Fn}) = \sum_{\mathbf{x} \in F} \mathbf{x}$$

where the summation is a vector sum. Notice that $\mathbf{a}_F / m_F$ is the center of gravity of F.

There are many slight variations on Chow parameters. They can be defined using $\{-1, +1\}$-logic instead of $\{0, 1\}$-logic or be defined such that

$$\mathbf{a}_F = \sum_{\mathbf{x} \in F} \mathbf{x} - \sum_{\mathbf{x} \in \bar{F}} \mathbf{x}.$$

In all cases the properties are similiar.

Definition 20. [Hu 1965] Sets $F, G \subseteq \mathbb{K}_n$ are said to be **equipollent** if and only if they have the same Chow parameters.

[5] These could have been called parameters, but Golomb calls them invariants. He uses them to classify similiar functions into groups, so the invariants are unchanged for a particular group.

Theorem 4. *[Chow 1961b] Let $F, G \subseteq \mathbb{K}_n$ be equipollent sets in $\mathbb{K}_n$. Then either both F and G are linearly separable and $F = G$ or both F and G are not linearly separable. In other words there is at most one linearly separable set for given Chow parameters.*

Proof. Let $F, G \subseteq \mathbb{K}_n$ be equipollent sets in $\mathbb{K}_n$. If either $F \subseteq G$ or $F \supseteq G$ then $m_F = m_G$ implies $F = G$. If $F \nsubseteq G$ and $F \not\supseteq G$ then $F \cap \bar{G} \neq \emptyset$ and $\bar{F} \cap G \neq \emptyset$, and $m_F = m_G$ implies that

$$|F \cap \bar{G}| = |F| - |F \cap G| = |G| - |F \cap G| = |\bar{F} \cap G|.$$

Also

$$\begin{aligned}
\sum_{\mathbf{X} \in F \cap \bar{G}} \mathbf{X} &= \sum_{\mathbf{X} \in F} \mathbf{X} - \sum_{\mathbf{X} \in F \cap G} \mathbf{X} \\
&= \mathbf{a}_F - \sum_{\mathbf{X} \in F \cap G} \mathbf{X} \\
&= \mathbf{a}_G - \sum_{\mathbf{X} \in F \cap G} \mathbf{X} \\
&= \sum_{\mathbf{X} \in G} \mathbf{X} - \sum_{\mathbf{X} \in F \cap G} \mathbf{X} \\
&= \sum_{\mathbf{X} \in \bar{F} \cap G} \mathbf{X}
\end{aligned}$$

Therefore F and G are $|F \cap \bar{G}|$-summable and by Theorem 1, F and G are not linearly separable. □

A Corollary of this theorem is that if 2 distinct sets are equipollent then neither set is linearly separable.

Theorem 5. *[Chow 1961b] Let $F \subseteq \mathbb{K}_n$ such that $(\mathbf{w}, t)$ with $\mathbf{w} \in \mathbb{R}^n$ and threshold $t \in \mathbb{R}$ is a separating system for F. The weights $\mathbf{w}$ are related to the Chow parameters of F, $(m, \mathbf{a})$ as follows:*

1. *If $a_i < m/2$ then $w_i < 0$.*
2. *If $a_i > m/2$ then $w_i > 0$.*
3. *If $a_i = m/2$ then F is independent of x_i ($w_i = 0$).*
4. *If $a_i > a_j$ then $w_i > w_j$.*
5. *If $a_i = a_j$ then there exists $(\mathbf{w}', t')$ with $w'_i = w'_j$ such that $(\mathbf{w}', t')$ realizes F.*

Theorem 6. *[Chow 1961b] Let $F \subseteq \mathbb{K}_n$ with $a_i < m/2$ for all $i \in \{1, 2, \ldots, n\}$. If $\mathbf{y} \in F$ and $\mathbf{x} \in \mathbb{K}_n$ such that $\mathbf{x} \prec \mathbf{y}$ then $\mathbf{x} \in F$.*

Theorem 7. *[Chow 1961b] Let $F, G \subseteq \mathbb{K}_n$. If $m_F = m_G$ and $a_{Fi} < m_F/2$ for all $i \in \{1, 2, \ldots, n\}$ and $\mathbf{a}_G \prec \mathbf{a}_F$ then F is not linearly separable.*

Proof. Suppose F is linearly separable. Then there exists a separating system $(\mathbf{w}, t)$ for F. By Theorem 5, $w_i < 0$ for $i \in \{1, 2, \ldots, n\}$. Since $\mathbf{a}_G \prec \mathbf{a}_F$, F and G are distinct. Hence $|F \cap \bar{G}| = |\bar{F} \cap G| = s$ where $s > 0$. For all $\mathbf{x} \in F$, $\mathbf{w} \cdot \mathbf{x} \geq t$ and for all $\mathbf{x} \in \bar{F}$, $\mathbf{w} \cdot \mathbf{x} < t$. This implies

$$\sum_{\mathbf{x} \in F \cap \bar{G}} \mathbf{w} \cdot \mathbf{x} \geq st > \sum_{\mathbf{x} \in \bar{F} \cap G} \mathbf{w} \cdot \mathbf{x}$$

Now we have

$$\begin{aligned}
\mathbf{a}_F \cdot \mathbf{w} &= \sum_{\mathbf{x} \in F} \mathbf{w} \cdot \mathbf{x} \\
&= \sum_{\mathbf{x} \in F \cap G} \mathbf{w} \cdot \mathbf{x} + \sum_{\mathbf{x} \in F \cap \bar{G}} \mathbf{w} \cdot \mathbf{x} \\
&> \sum_{\mathbf{x} \in F \cap G} \mathbf{w} \cdot \mathbf{x} + \sum_{\mathbf{x} \in \bar{F} \cap G} \mathbf{w} \cdot \mathbf{x} \\
&= \sum_{\mathbf{x} \in G} \mathbf{w} \cdot \mathbf{x} \\
&= \mathbf{a}_G \cdot \mathbf{w}
\end{aligned}$$

But $\mathbf{a}_G \prec \mathbf{a}_F$ and $w_i < 0$ for all $i \in \{1, 2, \ldots, n\}$ implies that $\mathbf{a}_G \cdot \mathbf{w} > \mathbf{a}_F \cdot \mathbf{w}$. This is a contradiction. Therefore F is not linearly separable. □

Lemma 1. *For a 2-monotonic Boolean function f of n variables with Chow parameters $(m, \mathbf{a})$, let $j, k \in \{1, 2, \ldots, n\}$ such that $j \neq k$ and let $C = \phi(1)\phi(2)\cdots\phi(n)$ be a cube with $cod(C) = 2$ defined by*

$$\phi(i) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i = k \\ * & \text{otherwise} \end{cases}$$

and let D be the cube opposite to C. Then $f_C \leq f_D$ if and only if $a_j \leq a_k$.

Winder [1971] makes an interesting observation (where he uses $\{-1, +1\}$-logic in only self-dual switching functions of $N = n + 1$ variables). When plotting N-tuples closely related to Chow parameters of all possible switching functions in N-space,

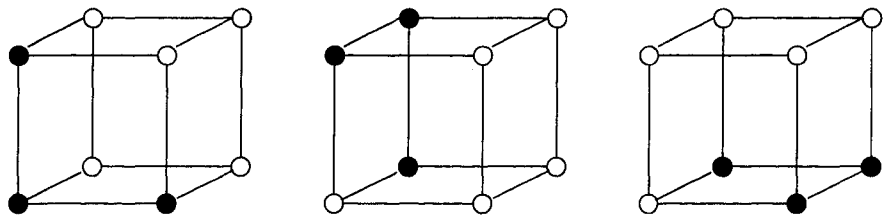

Figure 1.6: Equivalent sets $F_1, F_2, F_3 \subseteq \mathbb{K}_3$.

the N-tuples of threshold functions are generally further from the origin than non-threshold functions. Winder proves that there is a closed convex surface in N-space which exactly excludes the Chow parameters.

1.4 Equivalence Classes of Boolean Functions

As stated earlier the number of subsets of $\mathbb{K}_n$ is 2^{2^n}. Therefore finding all the linearly separable subsets of $\mathbb{K}_n$ by means of examining every subset quickly becomes infeasible ($\mathbb{K}_3$ has 256 subsets, $\mathbb{K}_4$ has 65536 subsets, $\mathbb{K}_5$ has 4294967296 subsets).

As it turns out, the subsets of $\mathbb{K}_n$ can be grouped into equivalence classes, where F and G are in the same equivalence class if there exists a complementation and/or permutation that converts F into G. Sets in the same equivalence class are said to be of the same type, Figure 1.6 depicts 3 subsets of $\mathbb{K}_3$ that are of the same type.

Definition 21. [Hu 1965] For a fixed $\mathbf{u} \in \mathbb{K}_n$, let the **u-complementation** $\gamma_{\mathbf{u}}\colon \mathbb{K}_n \rightarrow \mathbb{K}_n$ be defined so that $\gamma_{\mathbf{u}}(x_1x_2\cdots x_n) = y_1y_2\cdots y_n$ where

$$y_i = \begin{cases} x_i & \text{if } u_i = 0 \\ 1 - x_i & \text{if } u_i = 1. \end{cases}$$

n	1	2	3	4	5	6
N_n	3	6	22	402	1228158	400506806843728

Table 1.3: Number of NP-equivalent types of Boolean functions of n variables [Slepian 1953].

Definition 22. [Hu 1965] For a permutation of the integers $N = \{1, 2, \ldots, n\}$, $\sigma\colon N \to N$ the **permutation of variables** $\delta_\sigma\colon \mathbb{K}_n \to \mathbb{K}_n$ is defined

$$\delta_\sigma(x_1 x_2 \cdots x_n) = x_{\sigma(1)} \cdots x_{\sigma(n)}$$

Theorem 8. *[Hu 1965] Let $\alpha = \gamma_{\mathbf{u}} \circ \delta_\sigma\colon \mathbb{K}_n \to \mathbb{K}_n$ and let f be a Boolean function of n variables. If f is a threshold function then $f \circ \alpha$ and $\overline{f \circ \alpha}$ are also threshold functions. If f is k-monotonic then $f \circ \alpha$ and $\overline{f \circ \alpha}$ are k-monotonic. If f is k-summable then $f \circ \alpha$ and $\overline{f \circ \alpha}$ are k-summable.*

Definition 23. [Winder 1968][Muroga 1971] Boolean functions f and g of n variables are said to be **P-equivalent functions** if there exists a permutation of variables δ_σ such that $f \circ \delta_\sigma = g$. If there exists a permutation δ_σ and **u**-complementaion $\gamma_{\mathbf{u}}$ such that $f \circ \gamma_{\mathbf{u}} \circ \delta_\sigma = g$ then f and g are **NP-equivalent functions**. If there exists δ_σ and $\gamma_{\mathbf{u}}$ such that $f \circ \gamma_{\mathbf{u}} \circ \delta_\sigma = g$ or $f \circ \gamma_{\mathbf{u}} \circ \delta_\sigma = \bar{g}$ then f and g are said to be **NPN-equivalent functions**. Notice that this terminology is not related to that used in complexity of computation theory.

To find the properties of all elements of an equivalence class, only 1 representative needs to be examined. A method of calculating N_n, the number of NP-equivalent types of Boolean functions of n variables, is given by Slepian [1951] along with the number of types for $n \leq 6$, which are listed in Table 1.3.

Definition 24. A Boolean function f of n variables is **canonical** if the following properties hold on the Chow parameters $(m, \mathbf{a})$ of f:

$$a_1 \geq a_2 \geq \cdots \geq a_n \geq m/2 \quad \text{and} \quad m \leq 2^{n-1}.$$

It is well-known that every 2-monotonic Boolean function is NPN-equivalent to a unique canonical function. In particular, this is true for all threshold functions.

It has also been shown by Horváth [1994] that for a threshold functions f the invariance group[6] G of f is isomorphic to a direct product of symmetric groups.

SD Equivalence Classes

The NPN equivalence classes can be broadened with the inclusion of operations called **self-dualization** and **anti-self-dualization**.

Definition 25. [Goto and Takahasi 1963] Given a Boolean function f of n variables $(x_1, \ldots x_n)$, the **self-dualization** of f, f^{sd} is a $n+1$ variable Boolean function defined by

$$f^{sd} = x_0 f^d + \bar{x}_0 f.$$

Definition 26. For a self-dual function f of $n+1$ variables $x_0, \ldots x_n$, the **anti-self-dualization** of f is f such that $x_0 = 0$.

Definition 27. [Goto and Takahasi 1963] Boolean functions f and g belong to the same **SD equivalence class** if f coincides with g by any number of self-dualizations,

[6]The invariance group G of a Boolean function f is the set of permutations of the variables that leaves f unchanged.

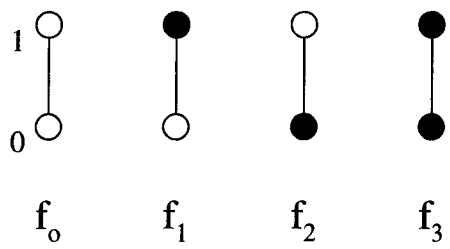

Figure 1.7: Boolean functions of $n = 1$ variable.

f	f^{sd}
$f_0(x_1) = 0$	$f_0^{sd}(x_0, x_1) = x_0$
$f_1(x_1) = x_1$	$f_1^{sd}(x_0, x_1) = x_1$
$f_2(x_1) = \bar{x}_1$	$f_2^{sd}(x_0, x_1) = \bar{x}_1$
$f_3(x_1) = 1$	$f_3^{sd}(x_0, x_1) = \bar{x}_0$

Table 1.4: Self-dualizations of the $n = 1$ variable Boolean functions.

anti-self-dualizations, negations of the function or the variables, and/or permutations of the variables.

Since self-dualization and anti-self-dualization change the number of variables, Booleans functions of a varying number of arguments are classified together. The $n = 1$ variable Boolean functions (see Table 1.1) are depicted in Figure 1.7. Figure 1.8 shows only the $n = 1$ variable Boolean functions divided into P, NP, NPN, and SD equivalence classes. At first it may not seem obvious that all the $n = 1$ variable Boolean functions belong in the same SD class. Inspection of the self-dualizations clearly show the equivalence of all the $n = 1$ variable Boolean functions. Table 1.4 lists the 2 variable self-dualizations of the functions and Figure 1.9 depicts them.

As indicated by Definition 27, the SD equivalence class containing f_0, f_1, f_2, f_3 also contains $f_0^{sd}, f_1^{sd}, f_2^{sd}, f_3^{sd}$ and all of the self-dualizations of the self-dualizations.

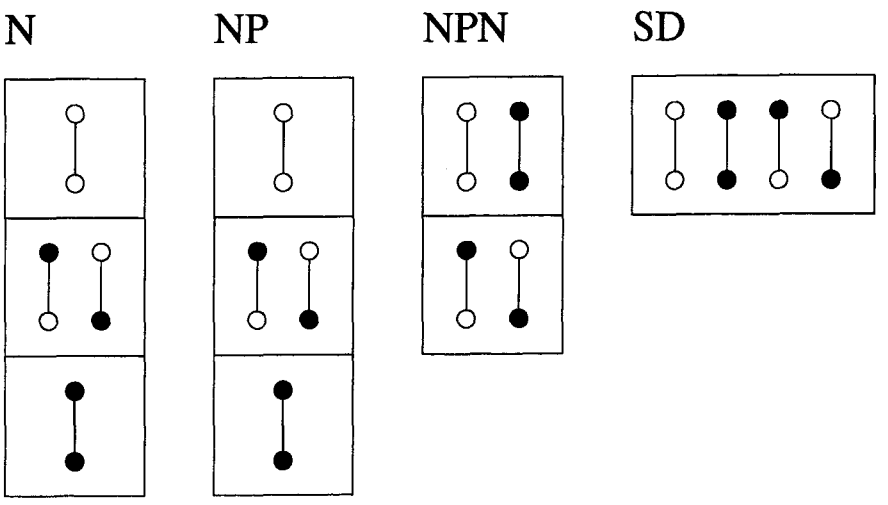

Figure 1.8: Boolean functions of $n = 1$ variable separated into P, NP, NPN, and SD equivalence classes.

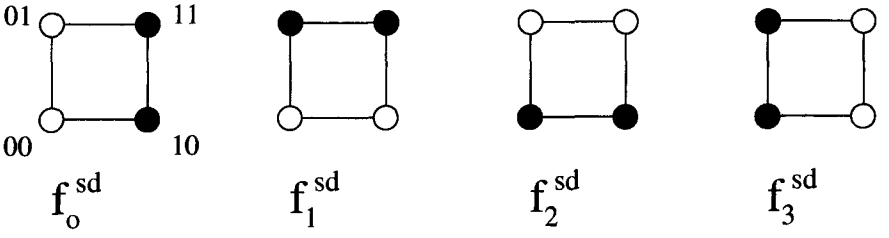

Figure 1.9: Self-dualizations of the $n = 1$ variable Boolean functions.

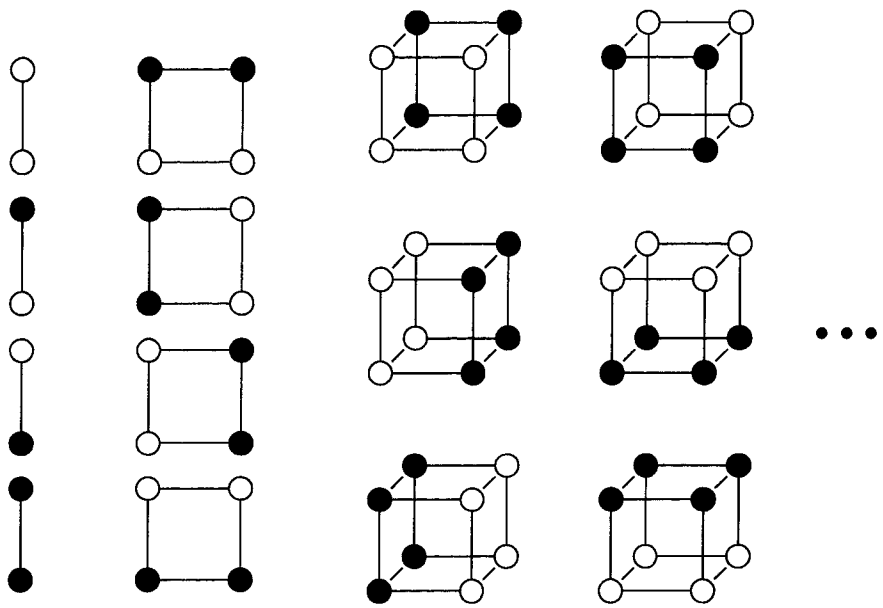

Figure 1.10: The SD class containing f_0, f_1, f_2, f_3.

Therefore, the SD equivalence class containing f_0, f_1, f_2, f_3 is the infinite class shown in Figure 1.10. Table 1.5 lists the number of P,NP,NPN, and SD equivalence classes containing Boolean functions of n variables [Muroga 1971]. Notice that the number of SD equivalence classes containing Boolean functions of n variables is far less than the number of Boolean functions of n variables. The following theorem states that either all or none of the functions in a particular SD equivalence class are threshold functions.

Theorem 9. *[Goto and Takahasi 1963] All Boolean functions belonging to the same self-dual class can be expressed with the same number of threshold functions.*

For example, if a Boolean function can be expressed using two threshold functions, then all the functions in the same self-dual class can be expressed using only

n	1	2	3	4	5
2^{2^n}	4	16	256	65536	4294967296
P	4	12	80	3984	37333248
NP	3	6	22	402	1228158
NPN	2	4	14	222	616126
SD	1	3	7	83	109958

Table 1.5: Number of P, NP, NPN, and SD equivalence classes containing Boolean functions of n variables [Muroga 1971].

two threshold functions. A corollary to this theorem is that if one function in a self-dual class is a threshold function, then all the functions in that self-dual class are threshold functions. Also, once a separating system is found for any of the threshold functions in a SD class, it is easy to obtain separating systems for the other functions in the class. For instance, if f is a threshold function of n variables with a separating system $(\mathbf{w}, t)$, then a separating system for f^d is $(\mathbf{w}, t^d)$ where $t^d = \sum_{i=1}^{n} w_i - t + 1$ and a separating system for f^{sd} is $(\mathbf{w}', t)$, where $\mathbf{w}' = (t - t^d, w_1 \cdots, w_n)$.

The conversion between functions of the same SD class is even simpler if we use $\{-1, 1\}$-logic instead of $\{0, 1\}$-logic. Let $f : \{-1, 1\}^n \mapsto \{-1, 1\}$ be a threshold function of n variables. If $(\mathbf{w}, t)$ is a separating system for f, then $(\mathbf{w}, -t)$ is a separating system for f^d and $(\mathbf{w}', 0)$ where $\mathbf{w}' = (-t, w_1 \cdots, w_n)$ is a separating system for f^{sd}. Thus for a logic gate that evaluates f, only the constant input needs to be changed for the gate to evaluate f^d and f^{sd}, see Figure 1.11. Considering that permutation and complementation of variables can be done outside of the gate, the same gate can be used to evaluate all the functions in an SD class (see [Muroga 1971]).

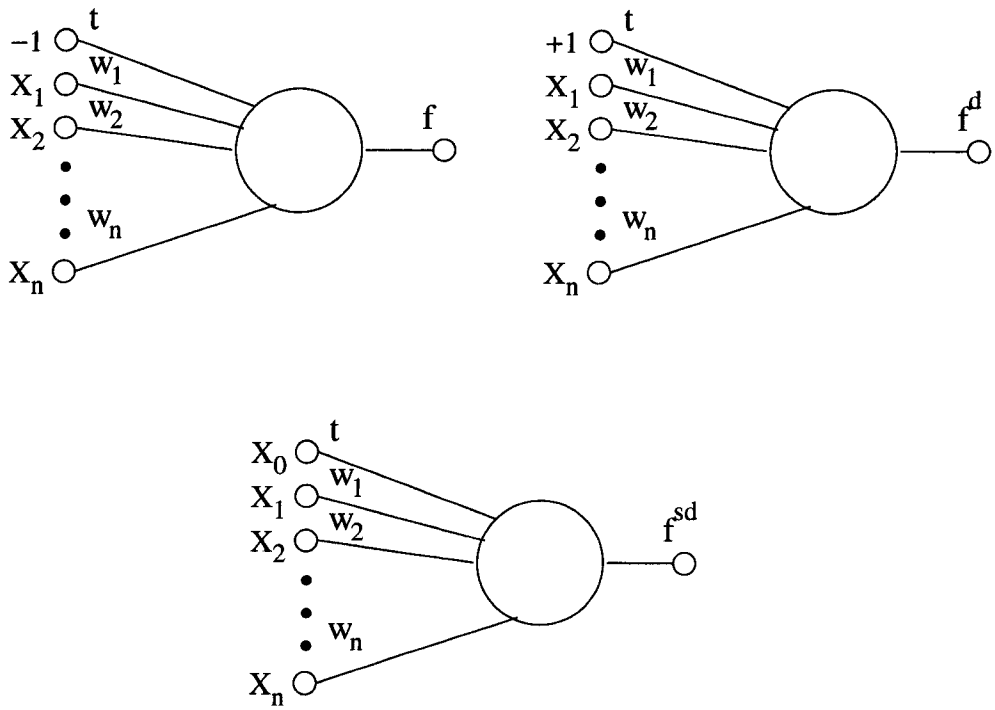

Figure 1.11: Logic gates evaluating f, f^d, and f^{sd} such that $f : \{-1,1\}^n \mapsto \{-1,1\}$.

1.4.1 Number of Self-dual Boolean Functions

It is known that the number of $n+1$ variable self-dual Boolean functions is equal to the number of n variable Boolean functions, and this fact is used in many papers. Goto and Takahasi[1962] prove that the self-dualization function is injective or one-to-one. It is easy to show that the self-dualization function is also surjective or onto.

Lemma 2. *The self-dualization function is onto.*

Proof. We will show that for every self-dual function g of $n+1$ variables (named $x_0, \ldots, x_n$) there exists a function of n variables $(x_1, x_2, \ldots, x_n)$ such that $g = f^{sd}$. Let f_1, f_2 be n variable functions such that $g = x_0 f_1 + \bar{x}_0 f_2$. Then $g^d = \overline{\bar{x}_0 f_1(\bar{x}_1\bar{x}_2\cdots\bar{x}_n) + x_0 f_2(\bar{x}_1\bar{x}_2\cdots\bar{x}_n)} = \overline{\bar{x}_0\overline{f_1^d} + x_0\overline{f_2^d}} = (\overline{\bar{x}_0\overline{f_1^d}})(\overline{x_0\overline{f_2^d}}) = (x_0 + f_1^d)(\bar{x}_0 + f_2^d) = x_0 f_2^d + \bar{x}_0 f_1^d$. Since $g = g^d$, we have $x_0 f_1 + \bar{x}_0 f_2 = x_0 f_2^d + \bar{x}_0 f_1^d$ which implies that $f_1 = f_2^d$. Then replacing f_1 by f_2^d we have $g = x_0 f_2^d + \bar{x}_0 f_2$. Thus g is a self-dualization of f_2. Therefore self-dualization is onto. □

It follows that the number of n variable Boolean functions is equal to the number of $n+1$ variable self-dual Boolean functions. Also since the self-dualization of a threshold function is also a threshold function, the number of n variable threshold functions is equal to the number of $n+1$ variable self-dual functions. It is of great advantage to work with $n+1$ variable self-dual functions instead of n variable functions, because there are fewer canonical $n+1$ variable self-dual functions than there are canonical n variable functions. We will consider this further in Chapter 2.

1.4.2 Generation of Canonical Sets

Since only the canonical sets are needed, there have been several attempts to generate only canonical sets for testing. Muroga et al [1962] give a method for obtaining canonical Boolean functions, using $(n-1)$ variable threshold functions to generate a canonical n variable function. Winder [1965] defines a lattice on the elements of $\mathbb{K}_n$ and uses it to generate only 2-monotonic self-dual canonical Boolean functions. Both these methods will be explored further in Chapter 2. In both papers, Boolean functions were tested for linear separability using linear programming and the realizing weight vectors for threshold functions are listed in tables according to the Chow parameters of a function. Muroga et al [1962] list all 6 variable canonical threshold functions and Winder [1965] list all 7 variable canonical threshold functions.

Ojha [2000] provides an enumeration of threshold functions with 3,4, and 5 variables using symmetry-adapted posets of hyperplane intersections as a method of generating functions. The author claims that his method only generates threshold functions, but has not proved any of his claims.

1.5 Classifying Threshold Functions

To determine whether a Boolean function is a threshold function, in other words, if the corresponding set is linearly separable, linear programming may be used. For a set $F \subseteq \mathbb{K}_n$, F is linearly separable if and only if the system of 2^n inequalities

$$(\mathbf{w}; t) \cdot (\mathbf{x}; -1) \geq 0 \qquad \text{for } \mathbf{x} \in F$$

$$(\mathbf{w}; t) \cdot (\mathbf{x}; -1) < 0 \qquad \text{for } \mathbf{x} \in \bar{F}$$

has a solution. The unknowns are t and the weights $\mathbf{w}$. If we introduce a sufficiently small $\epsilon > 0$ then the system of inequalities can be rewritten as

$$(\mathbf{w}; t, \epsilon) \cdot (\mathbf{x}; -1, -1) \geq 0 \qquad \text{for } \mathbf{x} \in F$$
$$(\mathbf{w}; t, \epsilon) \cdot (-\mathbf{x}; 1, -1) \geq 0 \qquad \text{for } \mathbf{x} \in \bar{F}.$$

A linear programming problem that maximizes $V(\epsilon) = \epsilon$ subject to the revised set of constraints has a positive solution if and only if the original system has a solution. This problem can be efficiently solved by using simplex algorithm or by Karmarker's algorithm [Keener 2004].

There have been many attempts to determine separability without using linear programming. Coates and Lewis [1961] describe a recursive method of determining separability. The test involves decomposing a Boolean function f so that $f = x_i f^i + \overline{x_i} f_i$. These new functions are then themselves decomposed, and their decompositions decomposed, resulting in a tree of functions, where each level has functions of a smaller number of variables than the previous level. The equation of the hyperplane is then found by starting at the bottom of the tree and traversing upwards. The drawback of this method is that if the resulting hyperplane does not separate f correctly then there are two far more complicated procedures to execute before it can be positively determined that the switching function f is not a threshold function. The authors suggest that for most functions of around 8 variables, 30 minutes of hand calculations should suffice, but calculations could take up to four hours. It would have taken the authors about 18 years to calculate the number of 8 variable threshold functions. Currently the number of 8 variable threshold functions

can be calculated in several minutes using linear programming on digital computers.

Even though linear programming is so efficient, decomposing a Boolean functions f about a variable is still employed as a means of determining linear separability [Picton 2001]. Every Boolean function f can be decomposed about a variable x_i such that $f = \bar{x}_i f_i + x_i f^i$, where f^i and f_i are the i-th upper and lower residues as defined in Definition 13. If f is a threshold function, it follows that f_i and f^i are also threshold functions, and can be realized with same weights, but different thresholds. If $(\mathbf{w}, t)$ is a separating system for f and $\mathbf{w}' = (w_1, \ldots, w_{i-1}, w_{i+1}, \ldots, w_n)$, then $(\mathbf{w}', t)$ is a separating system for f_i and $(\mathbf{w}', t - w_i)$ is a separating system for f^i [Picton 2001]. Picton states that if f_1 and f_2 are $(n-1)$ variable threshold functions with separating systems $(\mathbf{w}, t_1)$ and $(\mathbf{w}, t_2)$ then f_1 and f_2 can be combined to form a n variable threshold function where the weight for the extra variable is the difference between t_1 and t_2. It would be very nice if for every function $f = \bar{x}_i f_i + x_i f^i$, the separating systems (if they exist) of f_i and f^i could be used to find a separating system for f. Unfortunately there exist non-threshold functions for which both f_i and f^i are threshold functions. Also since every threshold function has an infinite number of separating systems, it is not likely that the separating systems found for f_i and f^i will have the same weights even if it is possible. Picton [2001] considers combining two $(n-1)$ variable functions which don't necessarily have the same weights. Picton presents some rules for this recombination, but has not yet provided rules for all cases.

For a class of threshold functions with positive integer weights that are less than or equal to t, denoted by C_t, it has been shown by Abboud et al [1999] that a threshold function f is identifiable as being of class C_t using an algorithm

that needs $n^{O(t^5)}$ membership queries, where a membership query returns $f(\mathbf{x})$ for $\mathbf{x} \in \mathbb{K}_n$. It was also shown that learning the class of threshold functions with weights from $\{-1, 0, 1\}$ requires at least $\Omega(2^n)$ membership queries. The problem of determining whether a function is a threshold function is known to be co-NP-complete [Garey and Johnson 1979], [Hegedüs and Megiddo 1996].

1.5.1 Classification Using Chow Parameters

Since Chow parameters define the sign and relative magnitude of the weights, it seems natural to use them as weights that realize the threshold function. Winder [1969] tests the accuracy of using Chow parameters as a weight vector of a threshold function. In the same paper Winder compares the accuracy of several methods whose weight vector is derived from the Chow parameters.

Winder defines $m[f]$ as the number of true vectors for a Boolean function f. According to Winder, the Chow parameters of a self-dual Boolean function f of n variables are defined as

$$P_i = \frac{1}{2}(m[f^i] - m[f_i]) \quad i \in \{1, 2, \ldots, n\}$$

where f^i and f_i are as given in Definition 13. Notice that this definition of Chow parameters is slightly different than Definition 19 given in Section 1.3. Winder tested 7 methods (some from literature and some new) on the 2470 canonical self-dual functions of $n = 8$ variables[7] presented in [Winder 1971]. Of the methods tested, the best results were given by a very complicated function termed the geometric rule,

[7]Recall from Section 1.4 that set of self-dual functions of $n+1$ variables is equivalent to the set of all functions of n variables.

Method	No. Correct/ No. Incorrect	Average No. Mistakes
Bayes	148/2322	12.24
Chow Parameters	185/2285	7.77
Dertouzos	664/1806	4.20
Cheb-Chow	597/1873	3.66
Sine	715/1755	3.61
Cos-Chow	740/1730	2.92
Geometric	810/1660	2.94

Table 1.6: Determining separability using 7 different methods [Winder 1971]. Column 2 lists the number of functions that were correctly classified as threshold functions, and Column 3 lists the average number of elements that differ between the actual function and the function calculated by the weight vector.

but it still only classified 810 out of the 2470 threshold functions correctly. No tests were performed on non-threshold functions. Table 1.6 lists the number of realized threshold functions and the average number of points $x \in \mathbb{K}_n$ that were incorrectly mapped in each case. Note that there does seem to be a difference between the Chow parameters actually used by Winder and the formula given for them which will be explored in further detail in Section 3.2.

Kaszerman [1963] tests for separability (realizability) by using the Chow parameters (he uses $\{-1, +1\}$-logic) as an approximation of a threshold realization. The center of gravity of the points which are classified incorrectly are then used to correct the original guess. Kaszerman proves that his algorithm converges to a realization for threshold functions, and for functions that are not separable, the resulting weight vector can be used as an approximation for the non-threshold function.

n	Number of Threshold Functions with n Variables, $N(n)$
1	4
2	14
3	104
4	1882
5	94572
6	15028134
7	8378070864
8	17561539552946

Table 1.7: Actual Numbers of Threshold Functions [Muroga 1971]

1.6 Number of Threshold Functions

Let $N(n)$ be the number of threshold functions with n variables. Table 1.7 lists exact numbers of threshold functions, or $N(n)$, for $n \leq 8$ as presented by Muroga [1971]. Muroga obtained his results using the set generation techniques described in Section 1.4 along with the simplex method. The exact number of threshold functions with more than 8 variables is not given in the literature.

Muroga has also found a lower bound for the number of threshold functions using the formula

$$(2^{n-1} + 1)N(n-1) < N(n)$$

(first proved by Yamija and Ibaraki [1965]) along with the value of $N(8)$ as determined by Muroga [1971]. A lower bound for $N(n)$ was found to be

$$2^{n(n-1)/2+32} < N(n) \text{ for } n \geq 8 \quad \text{[Muroga 1971]}.$$

The problem of counting threshold functions was studied as early as the 19th

century. Ludwig Schläfli, a Swiss mathematician alive during the 1800's, discovered that m hyperplanes in standard position separate $n-$dimensional Euclidean space into

$$2\sum_{i=0}^{n-1}\binom{m-1}{i}$$

cells. Schläfli [1850] used induction on dimension size to demonstrate his result (see [Anthony and Bartlett 1999]). Schläfli's result was used by Zuev [1989] to show that $N(n)$ is asymptotically less than 2^{n^2}, and then modified by Irmatov [1996] to show that $N(n)$ is asymptotically equal to

$$2\sum_{i=0}^{n}\binom{2^n-1}{i} = 2^{n^2-n\log_2 n+O(n)}.$$

Bounds on the Weights of Threshold Functions

It has been shown by Muroga [1971] that for a threshold function f of n variables, integer weights of size

$$|w_i| \leq 2^{-n}(n+1)^{(n+1)/2}$$

are sufficient to realize f. Håsted [1994] shows that for $n = 2^m$ for $m \geq 3$ there exists a function that requires weights

$$|w_i| \geq (1/2n)e^{(-4n^\beta)}2^{(n\log n)/(2-n)}$$

for all $i \in \{1, 2, \ldots, n\}$, where $\beta = \log_2 3/2$, which is essentially Muroga's upper bound. This means there is no room for improvement in Muroga's result.

Studies on Variables

Definitions for terms used in the following sections can be found in the literature cited. Hammer et al [2000] present an interesting study on the evaluation, strength and relevance of Boolean function variables. An evaluation function is defined as a ratio of the number of true vectors over the number of total vectors. For a given subset $S \subseteq \{1, 2, \ldots, n\}$ the evaluation function is used to find the relevance of the variables in S. It is found that the results are in strong agreement with classical strength of variables results.

Another study on the influence of variables of Boolean functions is presented by Kahn et al [1998]. In this paper the influence of a set S of variables is measured by the ability of S to make a function 0, given that the variables not in S are randomly chosen. The results presented here are closely related to distribution of Hamming distances given a family of binary vectors.

According to Saks [1993], the influence of a variable x_i on the Boolean function f is the fraction of the points $\mathbf{x} \in \mathbb{K}_n$ for which $f(\mathbf{x}) \neq f(x_1, \ldots, \bar{x}_i, \ldots x_n)$.

Specification Numbers

For a set H of Boolean functions of n variables and a sample set $X \subseteq \mathbb{K}_n$ $f, g \in H$ are **consistent** on X if $f(x) = g(x)$ for all $x \in X$. If the set X is such that for f and g to be consistent then f and g are the same function, then X is called a specifying sample for f. According to Anthony et al [1995], the **specification number** of f in H, $\sigma(f)$, is the cardinality of the smallest sample set X that specifies f. For the set H of threshold functions it is demonstrated that the upperbound of $\sigma(f), f \in H$ is 2^n (if f is constant) and the authors demonstrate that the lower bound of $\sigma(f)$ if

$n+1$. Also the average specification number of all threshold functions is calculated to be less than n^2.

1.6.1 Threshold numbers

The **threshold number** $t(f)$ of a positive Boolean function f of n variables is the least number of linear inequalities whose solution set on $0-1$ variables is the set of false vectors of f. Calculating $t(f)$ is difficult because it must be determined whether subsets of $f^{-1}(0)$ are linearly separable. An $O(n^3)$ algorithm is available that determines separability, but it is for a small class of functions that have prime implicants of only 2 variables [Hammer et al, 1981]. It has also been shown bu Zuev and Lipkin [1988] that the threshold numbers of almost all Boolean functions fall between the bounds $2^n/n \leq t(f) \leq \ln n 2^n/n$.

1.7 Applications of Threshold Functions

Threshold logic has many varied applications, including the modelling of logic gates [Muroga 1971] and magnetic cores [Coleman 1961]. Although threshold gates are not widely in use, current work is being done to provide low cost threshold devices [Quintana et al, 1997]. Also, Ramos et al [2003] present an optimized multi-operand adder using threshold logic.

Outside of logic gates, threshold functions are used for many other applications, most notably neural networks and simple games.

1.7.1 Artificial Neural Networks

An artificial neural network is a machine that emulates the complex biological neural network that is the human brain. The brain is composed of an interconnected set of approximately 10^{11} neurons, where each biological neuron has the complexity of a microprocessor. Artificial neural networks are simple abstractions of biological neural networks that are used to model the biological brain. Artificial neural networks are powerful machines that perform cognitive tasks such as speech recognition, face recognition and stock exchange share price predictions. For more details on artificial neural networks see [Hagan et al, 1996], [Anthony and Bartlett 1999] and [Sima and Orponen 2003]. The connection between Boolean functions and artificial neural networks is presented by Anthony [2003] and Picton [2000].

History of Neural Networks

Background work for neural networks occurred as early as the late 19th century. This interdisciplinary work in physics, psychology and neurophysiology emphasized general theories of learning, vision, and conditioning. In the 1940's Warren McCullogh and Walter Pitts showed networks could perform arithmetic or logical functions. Donald Hebb followed with a mechanism for learning in biological neurons. Practical applications of artificial neural networks started in the the late 1950s, with Frank Rosenblatt's perceptron network that performed pattern recognition. Other learning algorithms were introduced, but all suffered from inherent limitations. In the 1980s a backpropagation algorithm for multilayer neural networks by David Rumelhart and James McClelland answered some of the critcisms of the 1960s. This inspired new growth in the area of artificial neural networks.

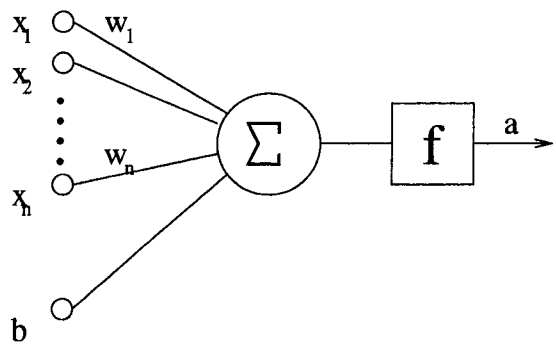

Figure 1.12: A Multiple-Input Neuron: $a = f(w_1x_1 + w_2x_2 + \cdots + w_nx_n - b)$.

Perceptrons

A perceptron neuron takes as input $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$. The neuron consists of weights $w_1, \ldots, w_n \in \mathbb{R}$, bias $b \in \mathbb{R}$ and a transfer function f. The weights emulate synapses in a biological neuron. The neuron output,

$$a = f(w_1x_1 + w_2x_2 + \cdots + w_nx_n - b)$$

corresponds to the signal a biological neuron would emit. Figure 1.12 shows a perceptron. There are several types of transfer functions that can be used for f. When f is

$$f(x) = sgn(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

then $a = sgn(w_1x_1 + w_2x_2 + \cdots + w_nx_n - b)$ is a linear threshold function, and the bias b is the threshold.

A Boolean function is called learnable if it can be calculated by a perceptron, and when the transfer function is sgn this is the same as determining if

a function is a threshold function. There are many papers and books that examine different methods of learning a function [Littlestone 1988], [Gallant 1990], [Schnelle and Engel 1991].

1.7.2 Simple Games

Hypergraphs or simple games have been studied for many years and lend themselves to many interpretations.

Definition 28. [Taylor and Zwicker 1999] A **hypergraph** G is a pair (P, W) in which P is a finite set and W is a collection of subsets of P. The elements of P are called **players** and the subsets in W are called **winning coalitions**.

Definition 29. [Taylor and Zwicker 1999] A hypergraph G satisfies **monotonicity** if $X \in W$ and $X \subseteq Y \subseteq P$ imply $Y \in W$. G is then called a **simple game**.

Definition 30. [Taylor and Zwicker 1999] A simple game $G = (P, W)$ is said to be **weighted** if there exists a **weight function** $w : P \mapsto \mathbb{R}$ and a **quota** $q \in \mathbb{R}$ such that $X \subseteq P$ is a winning coalition precisely when the sum of the weights of the players in X meets or exceeds quota.

Notice that a hypergraphs are a different way of looking at a Boolean functions and weighted simple games are just **b**-monotone threshold functions where $\mathbf{b} = (1, 1, \ldots, 1)$. Consider the Boolean function $f : \mathbb{K}_3 \mapsto \mathbb{K}$, defined by $f = x_1 + x_2x_3$. The corresponding hypergraph is $G = (P, W)$ with $P = \{1, 2, 3\}$ and $W = \{\{2, 3\}, \{1\}, \{1, 3\}, \{1, 2\}, \{1, 2, 3\}\}$. Notice that G satisfies monotonicity and thus is a simple game. If we know G is a simple game, we can write $W = \{\{2, 3\}, \{1\}\}$

and the rest is implied. It is important to note here that this definition of monotonicity is different than the definition of monotonic function in threshold theory.

Consider the simple game $G = (\{1,2,3,4\}, \{\{1,2\},\{3,4\}\}$, in this case the vectors $1100, 0011 \in \mathbb{K}_n$ are the corresponding true input vectors and $0101, 1010$ are false input vectors. It is easy to see that the corresponding Boolean function fails to be monotonic even though G satisfies monotonicity.

In simple game theory many of the concepts of threshold logic were redefined and rediscovered. Self-dual functions are now called constant-sum games, games that are based on k-assumable switching functions are called k-trade robust, and it is stated all weighted games are trade robust (k-trade robust for all $k \in \{1,2,\ldots,n\}$). Taylor and Zwicker [1995] determine that there does not exist some $k \geq 2$ for which k-trade robustness implies (k+1)-trade robustness. In simple game theory a 2-trade robust game is called a linear game (corresponding to 2-monotonic functions in threshold logic). Another name for trade robust games is locally weighted games and linear games are also called 2-locally weighted games. Unate Boolean functions correspond to 1-locally weighted games.

Much of the work of threshold logic has been repeated in simple game theory. For instance, Einy and Lehrer [1989] proved that linear games are not neccessarily weighted and Taylor and Zwicker [1995] rediscovered the Gableman examples. Work has also has been done to classify isomorphic games (see [Carreras and Freixas 1996], [Taylor and Zwicker 1999], and [Hammer et al, 2000]).

In simple game theory, much work is being done to study the influence that each variable of a Boolean function has on the output of the function. It is known that the average influence of a variable is $\Omega(1/n)$ although a Boolean function for which

each variable has the influence of $O(\log n/n)$ has been found by Ben-Or and Linial [1989] and the authors suggest that there is always 1 variable with greater influence than the others. This is an area that is also of interest to the field of distributed processing.

1.7.3 Convex Polytopes

For a d-dimensional cube C^d, a cut complex C is a sub-complex for which there exists a hyperplane that strictly separates the vertices of C from the rest of the d-cube. Lazarte [1989] gives a characterization for all the cut-complexes of the 5-cube. These are exactly the linearly separable Boolean functions of 5 variables and the algorithm for creating cut-complexes is far less efficient than the algorithm provided by Winder [1965]. For more information on convex polytopes see [Goodman and O'Rourke 2004] and [Grünbaum 2003].

1.7.4 Complexity of Boolean Functions

There is a topic of study called Complexity of Boolean Functions in which there seems to be no interest in whether or not a Boolean function is linearly separable. There is a different definition for a threshold function, where threshold functions of n variables, called T_k^n, are defined by

$$T_k^n(\mathbf{x}) = 1 \text{ if and only iff } x_1 + \cdots + x_n \geq k$$

for $\mathbf{x} \in \mathbb{K}_n$, $k \in \{1, 2, \ldots, n\}$. These types of functions are called majority functions, symmetry functions, or k out of n functions in threshold logic theory. For more

details on complexity of Boolean functions see [Dunne et al, 1995], [Wegener 1987], and [Newman and Wigderson 2002].

1.7.5 A Few More Applications

A threshold based system is a multi-class queueing system. For a system with k servers and n types of jobs, threshold functions can be used to activate or deactivate servers according to need. In the system described by Golubchik and Lui [2001] allocation and deallocation of servers to job classes are governed by a forward threshold vector and a reverse threshold vector. Many applications that could benefit from threshold based policies for resource management are given, as well as several issues that still need work, one of which is optimal settings of threshold values.

Swarm Intelligence is a distributed allocation algorithm modelled after natural systems such as ant colonies and bird flocks. Agassounon and Martinoli [2002] present a Swarm Intelligence approach based on threshold functions. Eui-Hong et al [1997] present a method for clustering of data (of interest in data mining applications) based on hypergraph models.

Since the applications of threshold logic are so far reaching, any new threshold logic results touch many areas of research.

Chapter 2

Canonical Set Generation

In Section 1.4.2 we discussed 2 different methods for generating canonical functions. In this chapter we will take a closer look at these methods. We will be reproducing the methods presented by Winder [1965] and Muroga et al [1962]. The objective of this chapter is to find the most efficient method of canonical threshold function generation. Descriptions of all algorithms used are presented in pseudocode, with corresponding programs written in C++. The source code for the programs is available by request.

2.1 Representation Of A Set

A subset $F \subseteq \mathbb{K}_n$ and its corresponding Boolean function f can be represented by a Boolean array of length 2^n. Each position in the array (starting with position 0 on the right) correlates to an element of $\mathbb{K}_n$. If there is a 1 in position i and if $\mathbf{x}$ in $\mathbb{K}_n$ is the binary representation of i, then $\mathbf{x}$ is in F. If there is a 0 in position i, then $\mathbf{x}$

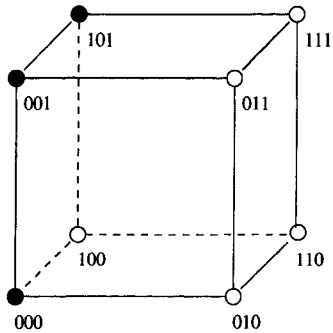

Figure 2.1: $A = \{000, 001, 101\} \subset \mathbb{K}_3$ is written 00100011.

is in $\bar{F}$.

The set $F = \{000, 001, 101\} \subseteq \mathbb{K}_3$ is depicted in Figure 2.1. The Boolean array representing F has a 1 in positions $0, 1, 5$ and thus is written 00100011. In our programs we are using the Standard Template Library[1] data structure bit_set to store the Boolean arrays. The bit_set data structure is designed to store arrays of Boolean characters and is very efficient.

2.2 Method of Muroga, Toda, and Kondo

Recall from Section 1.4.2 that Muroga et al [1962] used Boolean canonical threshold functions of $(n-1)$ variables to generate canonical threshold function candidates of n variables. This section includes a description of the method used by Muroga et al [1962] and the number of test cases it produces for different values of n.

Definition 31. [Muroga 1971] Let f be a Boolean function of n variables named $x_1 x_2 \cdots x_n$. Then $x_i \preccurlyeq x_j$ only if $f_D \leq f_C$ where f_C and f_D are restrictions of f over

[1]The Standard Template Library is accepted as part of the ANSI/ISO C++ draft standard.

opposite $(n-2)$-cubes C, D such that $C = \phi(1)\phi(2)\cdots\phi(n)$ with

$$\phi(k) = \begin{cases} 1 & \text{if } k = i \\ 0 & \text{if } k = j \\ * & \text{otherwise.} \end{cases}$$

If $f_D < f_C$ then $x_i \prec x_j$ and if both $x_i \preccurlyeq x_j$ and $x_j \preccurlyeq x_i$ then $x_i \approx x_j$. If either $x_i \preccurlyeq x_j$ or $x_i \succcurlyeq x_j$ then x_i and x_j are $\preccurlyeq$**-comparable**.

Muroga et al [1962] define a Boolean function f of n variables to be **canonical** if it satisfies the following 3 conditions:

1. f is positive.
2. The variables of f satisfy $x_1 \succcurlyeq x_2 \succcurlyeq \cdots \succcurlyeq x_n$.
3. f is such that $f \leq f^d$.

This definition is different than our previous definition of canonical functions.

Theorem 10. *[Muroga 1971] Let f be a Boolean function of n variables with Chow parameters $(m, \mathbf{a})$. If $x_i \prec x_j$, then $a_i > a_j$ and if $x_i \approx x_j$ then $a_i = a_j$. The converse is also true if x_i and x_j are $\preccurlyeq$-comparable.*

It is known that all unate functions have variables that are $\preccurlyeq$-comparable (see [Muroga 1971]). Non-unate functions exist that are canonical according to Definition 24, but have variables that are not $\preccurlyeq$-comparable and thus are not canonical according to Muroga's definition. For unate functions the two definitions are the same.

n	3	4	5	6	7	8	9
No. test cases	15	54	351	6001	460495	283034603	–

Table 2.1: Number of Boolean functions that need to be tested for linear separability using the method given by Muroga et al [1962].

As shown by Muroga et al [1962], if g, h are Boolean functions of (n-1) variables named $x_2, \ldots x_n$ such that the following conditions hold:

1. Both g and h are positive threshold functions of (n-1) variables $x_2, x_3, \ldots x_n$.
2. The variables of both g and h satisfy $x_2 \succeq x_3 \succeq \cdots \succeq x_n$.
3. For g^2, upper residue of g (see Definition 13), and h_2, lower residue of h, $g^2 \leq h_2$ holds.
4. $h \leq g^d$.

then $f = gx_1 + h$ is a canonical function of n variables as defined above.

Since both g and h are unate we can use the Chow parameters to check if the variables of g and h satisfy Condition 2. If we know that both g and h are threshold functions that satisfy canonical representative Conditions 1 and 2, then we only need to test for Conditions 3 and 4. All canonical threshold functions of n variables are generated using this method [Muroga et al, 1962]. Algorithm 1 provides the details.

Compared to Winder's method (as will be shown in the next section) this is an inefficient method of canonical Boolean function generation. Table 2.1 lists the number of Boolean functions that need to be tested for linear separablity using this method.

Algorithm 1 Count of the number of canonical threshold candidates using method of Muroga, Toda, and Kondo.

Require: *size* = number of canonical threshold functions of $(n-1)$ variables.
Require: *lower[size]* contains all canonical threshold functions of $(n-1)$ variables.

```
1: count = 0
2: for j = 0 to size do
3:   for k = 0 to size do
4:     g = lower[j]
5:     h = lower[k]
6:     gup = first upper residue of g
7:     hlow = first lower residue of h
8:     gdual = g^d
9:     if hlow ≤ gup and h ≤ gdual then
10:      Increment count
11:    end if
12:  end for
13: end for
```

2.3 Winder's Method

Winder [1965] presents a very good method for generating only 2-monotonic canonical self-dual Boolean functions of 8 variables. It is well known that the number of self-dual threshold $(n+1)$ variable functions is equivalent to the number of n variable threshold functions (see Section 1.4.1). In this section Winder's algorithm has been modified to work for self-dual Boolean functions of $(n+1)$ variables instead of self-dual Boolean functions of 8 variables. By using only self-dual canonical Boolean functions of $n+1$ variables, it is ensured that only one representative from each SD equivalence class is tested, and by only generating the 2-monotonic canonical Boolean functions the number of test cases needed is further reduced. Together these two aspects make Winder's method of set generation very efficient. This section describes Winder's method of Boolean function generation and provides the

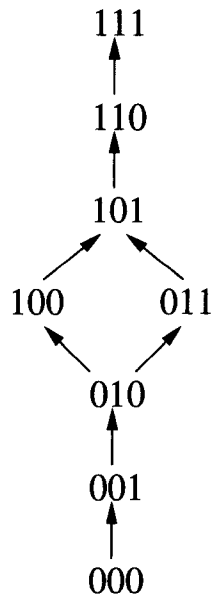

Figure 2.2: The lattice, L_2, defined by Winder [1965].

number of test cases it produces for different values of n.

Definition 32. [Winder 1965] For $\mathbf{x} \in \mathbb{K}_n$, $\dot{\mathbf{x}} \in \mathbb{N}^n$ is defined as $\dot{\mathbf{x}} = \dot{x_1}\dot{x_2}\cdots\dot{x_n}$ where

$$\dot{x_i} = \sum_{j=1}^{i} x_j \quad \text{for } i \in \{1, 2, \ldots, n\}.$$

Definition 33. [Winder 1965] For $\mathbf{x}, \mathbf{y} \in \mathbb{K}_n$, $\mathbf{x} \lessdot \mathbf{y}$ if $\dot{\mathbf{x}} \prec \dot{\mathbf{y}}$.

As an example, consider $\mathbf{x} = 010, \mathbf{y} = 100 \in \mathbb{K}_n$, then $\dot{\mathbf{x}} = (0, 1, 1)$ and $\dot{\mathbf{y}} = (1, 1, 1)$. Since $\dot{\mathbf{x}} \prec \dot{\mathbf{y}}$, $\mathbf{x} \lessdot \mathbf{y}$. The elements of $\mathbb{K}_n$ along with $\lessdot$ form a lattice[2], called L_2 [Winder 1965]. Figure 2.2 shows the lattice L_2 for $n = 3$.

Definition 34. [Winder 1965] A set $S \subseteq \mathbb{K}_n$ is **complete** when $\mathbf{x} \in S$ and $\mathbf{x} \lessdot \mathbf{y}$

[2]Winder uses the term "lattice" only for convenience. In fact, $\lessdot$ is a partial order, which is all that is required for Winder's argument.

implies $\mathbf{y} \in S$.

Theorem 11. *[Winder 1965] Let f be a Boolean function of n variables. Then f is 2-monotonic and canonical if and only if $F = f^{-1}(1)$ is complete.*

From now on we will call all 2-monotonic canonical Boolean functions **complete functions**.

Theorem 12. *[Winder 1965] For a Boolean function f of n variables $f^{sd} = x_0 f^d + \bar{x_0} f$ is a complete function of $n+1$ variables only if f is a complete function and $f^{sd}(1, 0, x_2, \ldots, x_n) \geq f^{sd}(0, 1, x_2, \ldots, x_n)$ for all pairs $(1, 0, x_2, \ldots, x_n), (0, 1, x_2, \ldots, x_n) \in \mathbb{K}_{n+1}$.*

Definition 35. A complete function f of n variables $x_1, \ldots, x_n$ is said to be **hypercomplete** if the self-dualization of f, $f^{sd} = x_0 f^d + \bar{x_0} f$, is a complete function of $n+1$ variables.

When working with self-dual functions it is only necessary to store the anti-self-dualization because the self-dual function is easily generated. Although Winder states he is generating complete self-dual functions of 8 variables, he is actually generating hypercomplete functions of 7 variables. Algorithm 2 describes the algorithm for generating hypercomplete functions of n variables. In Winder's code he uses an array of size 2^n which can hold the elements 0,1 or a blank to store his Boolean functions. We will be using two binary arrays (both of size 2^n), one to store the functions (called F) and another to show which elements can be modified (called V). Essentially if $V[\mathbf{x}] = 1$ then $F[\mathbf{x}]$ is still blank. Throughout the program we consider F and V as sets as well as arrays.

Algorithm 2 Counts the number of hypercomplete sets.

```
1: count = 0
2: F = ∅
3: V = 𝕂_n
4: push F, V onto a stack S
5: while S is non-empty do
6:   get F, V from the top of stack
7:   while V ≠ ∅ do
8:     x = the largest element in V
9:     push F, V onto S
10:    V[x] = 0
11:    for all y ⋖ x do
12:      V[y] = 0
13:    end for
14:  end while
15:  increment count {A test for linear separability could be performed here.}
16:  if S is non-empty then
17:    get F, V from the top of stack
18:    x = x_1 x_2 ··· x_n is the largest element in V
19:    F[x] = 1
20:    V[x] = 0
21:    if x_1 = 1 and x_2 = 1 then
22:      z = 10 x̄_3 x̄_4 ... x̄_n
23:      V[z] = 0
24:      for all y ⋖ z do
25:        V[y] = 0
26:      end for
27:    end if
28:    push F, V onto a stack S
29:  end if
30: end while
```

In Algorithm 2, lines 5-14 ensure that all the sets we create are complete, while lines 16-29 ensure that the sets are hypercomplete. We start with $F = \emptyset$, F is intially hypercomplete. V is initialized to $\mathbb{K}_n$. F, V are pushed onto an empty stack S. We will maintain V such that if $\mathbf{x} \in V$ and $\mathbf{x} \geq \mathbf{y}$ for all $\mathbf{y} \in V$, then $F \cup \{\mathbf{x}\}$ is hypercomplete. Initially $\mathbf{x} = (1, 1, \ldots, 1)$, and it is easy to see that $F = \{\mathbf{x}\}$ is hypercomplete. Essentially, for every F, V and $\mathbf{x}$ we have two options. The case where $\mathbf{x} \in F$ and the case where $\mathbf{x} \notin F$. Lines 10-13 deal with $\mathbf{x} \notin F$ pushing F, V back onto the stack to be retrieved again in line 17, where the case of $\mathbf{x} \in F$ will be taken care of.

In line 10, $\mathbf{x}$ is removed from V, and in lines 11 and 12 all the elements $\mathbf{y} \in V$ such that $\mathbf{y} \lessdot \mathbf{x}$ are removed. At this point every element in V can be added and F will be complete. Of course since we are not adding any points in this section, the condition for hypercompleteness has not been violated. The process continues through the while loop , each iteration pushing a new F, V on the stack, until V is empty. At this point (line 15), a test for separability could be performed.

In line 17, an F, V is retrieved from the top of the stack, and in line 18 the largest $\mathbf{x} \in V$ is found. Line 19 adds $\mathbf{x}$ to F and removes $\mathbf{x}$ from V. Lines 21-27 remove more elements from V to ensure the condition for hypercompleteness is met. A further explanation follows.

The condition of Theorem 12 can be restated as follows. If $f^{sd}(0, 1, x_2, \ldots, x_n) = 1$, then $f^{sd}(1, 0, x_2, \ldots, x_n) = 1$. Now, applying self-duality $f^{sd}(1, 0, x_2, \ldots, x_n) = (f^{sd})^d(1, 0, x_2, \ldots, x_n) = \bar{f^{sd}}(0, 1, \bar{x_3} \ldots, \bar{x_n})$. Therefore, the condition can be stated again in terms of just the functions f, or in terms of the set F. If $(1, x_2, \ldots, x_n) \in F$, then $(1, \bar{x_2}, \ldots \bar{x_n}) \notin F$. We can ensure this condition if every time we add an element

n	3	4	5	6	7	8	9
Hypercomplete	3	7	21	135	2470	319124	1214554343
Complete	6	15	62	591	20379	6154622	–

Table 2.2: Number of hypercomplete Boolean functions generated by Winder's algorithm compared to the number of complete Boolean functions.

$(1, x_2, \ldots, x_n)$ to F, we remove $(1, \bar{x_2}, \ldots, \bar{x_n})$ from V.

As it turns out, we only need to check when we are adding elements of the form $(1, 1, x_3, \ldots, x_n)$. If we are at the point of adding an element of the form $(1, 0, x_3, \ldots, x_n)$, we already know that $(1, 1, \bar{x_3}, \ldots, \bar{x_n})$ is not in F, because when $(1, 1, \bar{x_3}, \ldots, \bar{x_n})$ is added to F, $(1, 0, x_3, \ldots, x_n)$ is removed and thus cannot be added.

In Algorithm 2, line 21 checks if the element we just added is of the form $(1, 1, x_3, \ldots, x_n)$, and if so line 22 sets $\mathbf{z} = (1, 0, \bar{x_3}, \ldots, \bar{x_n})$. Then $\mathbf{z}$ is removed from V as well as all $\mathbf{y}$ such that $\mathbf{y} \lessdot \mathbf{z}$.

This method is a very efficient method of generating test cases. To generate all the canonical functions of 9 variables took about 7 hours[3]. As testing for separability is fairly time intensive, it is a lengthy procedure to determine which of the canonical 9 variable functions generated are threshold functions. A drawback to this algorithm is that it is not easily adaptable to run on parallel machines. Table 2.2 lists the number of hypercomplete functions generated. Just for comparison, Table 2.2 also list the number of complete functions of n variables.

[3]The C++ programs were run on quad 550MHz Sun SPARC IIi processors.

Chapter 3

Classifying Threshold Functions

In Section 1.5 we discussed many methods for determining whether a given Boolean function is a threshold function. Conventionally a Boolean function is converted into a system of inequalities which are solved by means of linear programming. Many authors ([Muroga 1971], [Winder 1965], and [Hu 1965]) provide details of such a conversion. In Section 3.1.1 we will describe in detail an efficient method of converting a canonical Boolean function into a system of inequalities [Hu 1965]. In Section 3.1.2 we give the formula for calculating the order of an equivalence class containing a particular function.

In the sections following we will explore several methods of determining whether a function is a threshold function. Section 3.2 discusses methods where a weight vector is derived from the Chow parameters while Section 3.3 investigates another method of deriving a weight vector.

3.1 Number of Threshold Functions

3.1.1 Simplex Method and Separability

In Section 1.5 we described how a system of linear inequalities can be used to determine if a Boolean function is a threshold function. If it is known that a set is canonical, then the number of inequalities can be reduced by exploiting the characteristics of canonical sets.

Definition 36. Let $F \subseteq \mathbb{K}_n$. An element $\mathbf{x} \in \mathbb{K}_n$ is a **boundary point** of F if for all $\mathbf{y} \in \mathbb{K}_n$ such that $\mathbf{x} \lessdot \mathbf{y}$, $\mathbf{y} \in F$ and for all $\mathbf{w} \in \mathbb{K}_n$ such that $\mathbf{w} \lessdot \mathbf{x}$, $\mathbf{w} \in \bar{F}$.

For a canonical set $F \subseteq \mathbb{K}_n$ we can create a system of inequalities which we can use to find a separating system $(\mathbf{w}, t)$ for F. We will be using the dual simplex method to maximize the expression $y = t + w_1 + w_2 + \cdots + w_n$ subject to the following constraints. For every boundary point of F, $\mathbf{x} \in \mathbb{K}_n$, if $\mathbf{x} \in F$ then 3.1 is a constraint and if $\mathbf{x} \in \bar{F}$ then 3.2 is a constraint.

$$-t + w_1x_1 + w_2x_2 + \cdots + w_nx_n \geq 0 \tag{3.1}$$

$$-1 - t - w_1x_1 - w_2x_2 - \cdots - w_nx_n \geq 0 \tag{3.2}$$

We also need to add the constraints

$$w_1 - w_2 \geq 0$$
$$w_2 - w_3 \geq 0$$
$$\vdots$$
$$w_{n-1} - w_n \geq 0.$$

The solution of dual simplex method is known to be optimal when the first element of each row of the simplex tableau is greater than zero. Dual simplex method iterates while there exists a row that starts with a negative number. If such a row is found that has no positive entries, then there is no solution and iteration halts. For more information see [Hu 1965] and [Winder 1965]. As an example, consider the canonical set $F \subseteq \mathbb{K}_5$ defined by the array

$$11111100100000001000000000000000.$$

The boundary points $\mathbf{x} \in \mathbb{K}_n$ for F such that $\mathbf{x} \in F$ are 01111 and 11010, while the boundary points $\mathbf{x} \in \bar{F}$ are 10110 and 11001. In this case, the programming

problem to be solved is as follows:

$$y = t+w_1 + w_2+w_3 + w_4+w_5 \qquad \text{Maximize } y \text{ subject to the following constraints.}$$

$$\begin{aligned}
-t \quad &+ w_2+w_3 + w_4+w_5 \geq 0 \\
-t+w_1 + w_2 \quad &+ w_4 \quad \geq 0 \\
-1-t-w_1 \quad &-w_3 - w_4 \quad \geq 0 \\
-1-t-w_1 - w_2 \quad &-w_5 \geq 0 \\
w_1 - w_2 \quad &\geq 0 \\
w_2 \ -w_3 \quad &\geq 0 \\
w_3 - w_4 \quad &\geq 0 \\
w_4 \ -w_5 &\geq 0
\end{aligned}$$

When this system is solved using the dual simplex method, as described in [Hu 1965], the separating system ((3,3,2,2,1),8) is found.

In Section 2.3, a method to generate all of the hypercomplete subsets of $\mathbb{K}_n$ was presented. Upon generation of a hypercomplete set we may use linear programming to calculate whether the set is linearly separable. Table 3.1 lists the number of hypercomplete subsets of $\mathbb{K}_n$ for $n \leq 9$. In the literature the number of hypercomplete functions can be found for $n \leq 8$, and it is believed the result for $n = 9$ is a new result.

n	3	4	5	6	7	8	9
$HN(n)$	3	7	21	135	2470	175428	52980624

Table 3.1: Number of hypercomplete threshold functions of n variables, denoted by $HN(n)$.

3.1.2 Number of Threshold Functions

In this section we will explain how to calculate the number of all threshold functions knowing only the canonical threshold functions. We need to find out how many Boolean functions are in the equivalence class of a canonical function. Any permutation and/or complementation of variables results in an equivalent Boolean function. Let $f : \mathbb{K}_n \rightarrow \mathbb{K}$. Then the number of permutations is $n!$. Of course if f is symmetric in 2 or more variables then the permutation of those variables results in f itself (see Theorem 5). Since symmetric variables result in the Chow parameters having equal a_i the number of permutations of a f is the number of permutations of the Chow parameters of f. The number of complementations of variables of f is 2^n, but if a function is independent of a variable then complementing that variable results in f. By Theorem 5 we know that a function is independent of a variable x_i if and only if the Chow parameter $a_i = m/2$. Therefore if d is the number of $a_i \neq m/2$, then the number of complementations of variables resulting in functions different from f is 2^d.

For $f : \mathbb{K}_n \rightarrow \mathbb{K}$ with Chow parameters $(m, \mathbf{a})$, let p be the number of permutations of $\mathbf{a}$, and let d be the number of $a_i \neq m/2$. Then the number of Boolean functions in the same equivalence class as f is $2^d p$.

During the process of calculating the number of hypercomplete threshold func-

n	Number of Threshold Functions with n Variables, $N(n)$
3	104
4	1882
5	94572
6	15028134
7	8378070864
8	17561539552946
9	144130531453121108

Table 3.2: Numbers of threshold functions of n variables.

tions, the number of equivalent Boolean functions for each hypercomplete function was calculated, therefore determining the total number of threshold function of n variables. The results are listed in Table 3.2. For $n \leq 8$ our results match that of the literature. To our knowledge, the number of threshold functions of 9 variables is not given in the literature, so it is believed that this is a new result.

3.2 Chow Parameters and Separability

As briefly discussed in Section 1.5.1, a comparison of 7 methods for deriving an approximate realization for a threshold function given its Chow parameters are presented by Winder [1969]. The methods are all tested on the 2470 canonical self-dual threshold functions of 8 variables. Winder defines Chow parameters as n parameters named $P_1, \ldots, P_n$ where $P_i = \frac{1}{2}(m[f_{x_i}] - m[f_{\bar{x}_i}])$, $m[f]$ is the number of $\mathbf{x} \in \mathbb{K}_n$ such that $f(\mathbf{x}) = 1$, and f_{x_i} and $f_{\bar{x}_i}$ are the functions of $n-1$ variables obtained from f by setting $x_i = 1$ or $x_i = 0$. Each method tested employed a weight function Φ that assigned a weight to each Chow parameter, $w_i = \Phi(P_i)$. The first method given by Winder [1969] is called *Chow Parameters* and the weight function

given is $\Phi(P_i) = P_i$, but in a table listing w_i for each possible P_i there are very different values given as the weights used. Table 3.3 lists a portion of the weights used by Winder [1969]. Winder states that using the Chow parameters as weights realizes 185 of the 2470 threshold functions, but these may be incorrect results. In the next section, it will be shown that using the Chow parameters in the weight vector produces much better results than those reported by Winder.

3.2.1 Derivation of Chow Parameters

There are many slight variations for calculating Chow parameters, and although the properties of each variation are similar, using the different types as a separating weight vector produces significantly different success rates. In this section, we use two accepted definitions of Chow parameters as weight vectors and test how many threshold functions are correctly separated. Once the weight vector $\mathbf{w}$ is calculated for $F \subseteq \mathbb{K}_n$ we test all $\mathbf{x} \in \mathbb{K}_n$ to see if there exists $t \in \mathbb{R}$ such that $\mathbf{w} \cdot \mathbf{x} \geq t$ for all $\mathbf{x} \in F$ and $\mathbf{w} \cdot \mathbf{x} < t$ for all $\mathbf{x} \in \bar{F}$. Algorithm 3 describes how we determine if there exists a threshold value. We are using the set of hypercomplete Boolean subsets of $\mathbb{K}_n$ as our testcases. Derivation 1 and Derivation 2 are distinct methods of calculating Chow parameters. Derivation 1 uses Definition 19, and Derivation 2 uses the definition of Chow parameters given by Winder [1969].

Chow Parameters	Weights
P_i	$w_i = \Phi(P_i)$
0	0
1	31
2	63
3	94
4	125
5	156
6	188
7	219
8	250
9	281
10	313
⋮	⋮
55	1719
56	1750
57	1781
58	1813
59	1844
60	1875
61	1906
62	1938
63	1969
64	2000

Table 3.3: Excerpt of the table given by Winder [1969].

Algorithm 3 Determine the existence of a threshold t for a given weight vector $\mathbf{w}$.

Require: $F \subseteq \mathbb{K}_n$ is a hypercomplete Boolean set
Require: $\mathbf{w} \in \mathbb{R}^n$
1: $t = n * 2^n$ {This is greater than $\mathbf{w} \cdot \mathbf{x}$ for all $\mathbf{x} \in \mathbb{K}_n$.}
2: $u = 0$
3: **for** $\mathbf{x} \in \mathbb{K}_n$ **do**
4: $d = \mathbf{w} \cdot \mathbf{x}$
5: **if** $\mathbf{x} \in F$ **then**
6: **if** $d < t$ **then**
7: $t = d$
8: **end if**
9: **else**
10: **if** $d > u$ **then**
11: $u = d$
12: **end if**
13: **end if**
14: **end for**
15: **if** $t > u$ **then**
16: $(\mathbf{w}, t)$ is a separating system for F
17: **else**
18: There does not exist t such that (w, t) realizes F
19: **end if**

Derivation 1

Let F be a hypercomplete set $F \subseteq \mathbb{K}_n$ with Chow parameters $(m, \mathbf{a})$. Let a weight vector $\mathbf{w}$ be $w_1 w_2 \cdots w_n$, where

$$w_i = a_i \text{ for all } i \in \{1, 2, \ldots, n\}$$

The number of functions that are correctly identified as threshold functions using this weight vector is listed in Table 3.4 under D1.

Derivation 2

Let F be a hypercomplete set $F \subseteq \mathbb{K}_n$ with Chow parameters $(m, \mathbf{a})$. Let a weight vector $\mathbf{w}$ be $w_1 w_2 \cdots w_n$, where

$$w_i = 2a_i - m \text{ for all } i \in \{1, 2, \ldots, n\}$$

This adjustment is consistent with using $\{-1, 1\}$-logic instead of $\{0, 1\}$-logic or calculating the Chow parameters such that

$$\mathbf{a}_F = \sum_{\mathbf{x} \in F} \mathbf{x} - \sum_{\mathbf{x} \in \bar{F}} \mathbf{x}.$$

The number of functions that are correctly identified as threshold functions using this weight vector is listed in Table 3.4 under D2, and as shown this produces a much better weight vector than the first derivation. This derivation is the method first given by Winder [1969], but instead of realizing 185 functions, we correctly

n	3	4	5	6	7	8	9
$N(n)$	3	7	21	135	2470	175428	52980624
D1	3	6	12	37	123	1236	–
D2	3	7	20	115	1405	37221	–

Table 3.4: Number of threshold functions that are separated correctly when using Chow parameters as a weight vector. $N(n)$ is the number of hypercomplete threshold functions of n variables.

realize 1405 threshold functions.

3.2.2 Adaptations of Chow Parameters

In the previous section we used the Chow parameters without modification as a weight vector. In this section, we use a function of the Chow parameters as a weight vector and test how many hypercomplete threshold functions are correctly separated.

Derivation 3

Inspection of the optimal weight vectors given by simplex method, shows that the weights are usually small integers. For this derivation of a weight vector we use the Chow parameters as a guide when choosing weights for a weight vector.

Let F be the hypercomplete set $F \subseteq \mathbb{K}_n$ with Chow parameters $(m, \mathbf{a})$. Let a weight vector $\mathbf{w}$ be $w_1 w_2 \cdots w_n$ where

$$w_1 = \begin{cases} 0 & \text{if } a_1 \text{ equals } m/2 \\ 1 & \text{otherwise} \end{cases}$$

and for all $2 \leq i \leq n$

$$w_i = \begin{cases} w_{i-1} & \text{if } a_i \text{ equals } a_{i-1} \\ w_{i-1} + 1 & \text{otherwise} \end{cases}$$

The number of functions that are correctly identified as threshold functions using this weight vector is listed in Table 3.5 under D3. Using the modification of Derivation 2 combined with this derivation yields the same results. This is because the modification of Derivation 2 on the Chow parameters preserves the relative magnitudes of $\mathbf{a}$.

Derivation 4

In Derivation 3, for a hypercomplete $F \subseteq \mathbb{K}_n$ with Chow parameters $(m, \mathbf{a})$ whenever $a_i > a_{i+1}$ we set $w_i = w_{i-1} + 1$. In this derivation, the weight vector is improved by increasing the amount added to w_{i-1} in the following cases. Inspection of the optimal weight vectors showed that when $a_i > a_{i-1}$ and $a_i \geq m$ then $w_i > w_{i-1} + 1$. In this trial, the weight vector is adjusted accordingly. Let F be a hypercomplete set $F \subseteq \mathbb{K}_n$ with Chow parameters $(m, \mathbf{a})$. Let a weight vector $\mathbf{w}$ be $w_1 w_2 \cdots w_n$, where

$$w_1 = \begin{cases} 0 & \text{if } a_1 \text{ equals } m/2 \\ 1 & \text{otherwise} \end{cases}$$

and for all $2 \leq i \leq n$ set j such that

$$j = \begin{cases} 2 & \text{if } a_i \geq m \\ 1 & \text{otherwise} \end{cases}$$

and

$$w_i = \begin{cases} w_{i-1} & \text{if } a_i \text{ equals } a_{i-1} \\ w_{i-1} + j & \text{otherwise} \end{cases}$$

The number of functions that are correctly identified as threshold functions using this weight vector is listed in Table 3.5 under D4. This modification does increase the number of correctly classified threshold functions and all the correctly classified functions of trial D3 are also classified correctly in D4. It would be interesting to know if all the functions classified correctly by D3 are also classified correctly by D4 for Boolean functions of $n > 8$ variables. This could be a subject of future study.

Derivation 5

In this section we try a different adjustment to the weight vector found in derivation 3. Let F be a hypercomplete set $F \subseteq \mathbb{K}_n$ with Chow parameters $(m, \mathbf{a})$. Now adjust $\mathbf{a}$ such that $a_i = 2a_i - m$. Let a weight vector $\mathbf{w}$ be $w_1 w_2 \cdots w_n$, where

$$w_1 = \begin{cases} 0 & \text{if } a_1 \text{ equals } m/2 \\ 1 & \text{otherwise} \end{cases}$$

n	3	4	5	6	7	8	9
$N(n)$	3	7	21	135	2470	175428	52980624
D3	3	7	20	74	267	872	–
D4	3	7	21	91	329	1037	–
D5	3	7	20	84	317	1120	–
D4 or D5	3	7	21	105	451	1681	–

Table 3.5: Number of threshold functions that are separated correctly when using functions of Chow parameters as a weight vector. $N(n)$ is the number of hypercomplete threshold functions of n variables.

and for all $2 \leq i \leq n$ set j such that

$$j = \begin{cases} 2 & \text{if } a_i \geq r a_{i-1} \text{ where } r \text{ is the multiplicity of } a_{i-1} \text{ in } \mathbf{a} \\ 1 & \text{otherwise} \end{cases}$$

and

$$w_i = \begin{cases} w_{i-1} & \text{if } a_i \text{ equals } a_{i-1} \\ w_{i-1} + j & \text{otherwise} \end{cases}$$

The number of functions that are correctly identified as threshold functions using this weight vector is listed in Table 3.5 under D5. This modification does classify more threshold functions correctly than Derivation 3, and for some dimensions more than Derivation 4. Derivation 4 and 5 classify different functions correctly and the number of threshold functions classified correctly by Derivation 4 or Derivation 5 is also listed in Table 3.5.

3.3 Border Sums and Separability

In the previous section we tested weight vectors derived from Chow parameters. Consequently, all of our conjectured weight vectors were derived from the sum of all elements of a set $F \subseteq \mathbb{K}_n$. When determining the weight vector using the simplex method, only the boundary points are needed for canonical threshold functions. In this section we define sums of subsets of F as weight vectors and test whether the F is separated correctly. The number of boundary points is very small and the sum of the boundary points yields a very poor weight vector. Here we define points that are related to 1-monotonic (unate) functions in the same manner that boundary points are related to 2-monotonic functions.

Definition 37. Let $F \subseteq \mathbb{K}_n$. An element $\mathbf{x} \in \mathbb{K}_n$ is a **border point** of F if for all $\mathbf{y} \in \mathbb{K}_n$ such that $\mathbf{x} \prec \mathbf{y}$, $\mathbf{y} \in F$ and for all $\mathbf{w} \in \mathbb{K}_n$ such that $\mathbf{w} \prec \mathbf{x}$, $\mathbf{w} \in \bar{F}$.

Derivation 6

Let F be a hypercomplete set $F \subseteq \mathbb{K}_n$ and let $B(F) = \{\mathbf{x} \in F | \mathbf{x}$ is a border point of $F\}$ Let a weight vector $\mathbf{w}$ be

$$\mathbf{w} = \sum_{\mathbf{x} \in B(F)} \mathbf{x}.$$

The number of functions that are correctly identified as threshold functions using this weight vector is listed in Table 3.4 under D6. Although this method does not yield very good results, it should be considered as a starting point for future research. One idea for modification is to include points adjacent to the border points to contribute to the weight vector.

n	3	4	5	6	7	8	9
$N(n)$	3	7	21	135	2470	175428	52980624
D6	3	7	14	43	103	679	–

Table 3.6: Number of threshold functions that are separated correctly when using border sums to derive weight vectors.

Chapter 4

Conclusion

The field of Boolean algebra is not very standardized. Since Boolean algebra is the basis of so many topics (neural networks, coding theory, distributed algorithms...), much of the work with threshold functions is done using different terminology and notation. In Chapter 1, I have brought together many theorems of threshold functions and unified them with a common notation. In some cases, work has been duplicated in different fields.

4.1 Generation of Boolean Functions

In Chapter 2, two methods of Boolean function generation are duplicated. The method presented by Winder [1965] is a very efficient method of function generation. Winder's method is so efficient because it only generates hypercomplete Boolean functions, which reduces the number of functions due to the fact that only one representative from a SD class is generated. Generating only hypercomplete

functions ensures that every function generated is at least 2-monotonic, which again reduces the number of test cases generated. A drawback of Winder's algorithm is that it is not very intuitive. The algorithm for set generation presented by Muroga et al [1962] is far more natural as it builds an n variable function using two $n-1$ variable functions. Muroga's algorithm generates only canonical functions, giving one representative from every NPN class. Muroga's algorithm is very poor compared to Winder's algorithm. Future work could include trying to find a new algorithm that builds n variable hypercomplete functions from $n-1$ hypercomplete functions. Building n variable functions from $n-1$ variable functions presents some benefits, one of which is that the exact number of iterations can be easily determined, another is that the work could be easily adapted to run on parallel machines.

4.2 Number of Threshold Functions

The exact number of threshold functions of $n \leq 8$ variables, $N(n)$, has been given in Muroga [1971] and Picton [2000]. Using the algorithm presented by Winder [1965] for set generation and the dual simplex method to determine separability, we have found the number of threshold functions of 9 variables to be 144130531453121108. Using Winder's algorithm, 1214554343 hypercomplete Boolean functions were generated. Of these 1214554343 hypercomplete Boolean functions, 52980624 were found to be threshold functions. Using the techniques described in Section 3.1.2, the total number of 9 variable threshold functions was found from the self-dualizations of the 52980624 hypercomplete Boolean functions. To our knowledge, the number of threshold functions of 9 variables is not given in the literature, so it is believed that

this is a new result.

4.3 Chow Parameters and Linear Separability

Theorem 5 shows that the Chow parameters and the weights that realize a threshold function are closely related. Therefore, it is natural to try to use the Chow parameters as the weight vector; Winder [1969] tested how many of the 2470 complete self-dual threshold functions of 8 variables were separated correctly. It was determined that only 185 functions were correctly seperated. In Chapter 3, we show that using the Chow parameters as a weight vector yields much better results than presented by Winder. We found that the Chow parameters correctly separated 1405 of the 2470 complete self-dual threshold functions of 8 variables.

In Section 3.2.2 we developed methods of determining a weight vector from the Chow parameters, none yielding better separation than the Chow parameters alone. Although using linear programming to determine separability is efficient, it doesn't provide any insight as to what characterizes a threshold function. If an intuitive method for determining separability could be found, then ,perhaps, a formula that gives the number of threshold functions based on n would follow.

4.4 Border and Boundary Points

In Section 3.3, border points were defined, and a separability test using border points was developed. Although the separability test was very poor, it is likely that border and boundary points are very important. I would like to test out more

methods derived from border and boundary points, since each border or boundary point contains a lot of information about other points. In the future I would like to try out weighted sums based on the number of points a border or boundary point implies, or look at the Hamming distances between boundary points.

Bibliography

[Abboud et al, 1999] Elias Abboud, Nader Agha, Nader H. Bshouty, Nizar Radwan, and Fathi Saleh. 1999. Learning threshold functions with small weights using membership queries. *Annual Conference on Computational Learning Theory.*

[Agassounon and Martinoli 2002] William Agassounon and Alcherio Martinoli. 2002. Efficiency and robustness of threshold-based distributed allocation algorithms in multi-agent systems. *AAMAS*, pages 1090–1097.

[Anthony and Bartlett 1999] Martin Anthony and Peter Bartlett. 1999. *Neural Network Learning: Theoretical Foundations.* Cambridge University Press, Cambridge, UK.

[Anthony et al, 1995] Mark Anthony, Graham Brightwell, and John Shawe-Taylor. 1995. On specifying boolean functions by labelled examples. *Discrete Mathematics.*

[Anthony 2003] Martin Anthony. 2003. Boolean functions and artificial neural networks. Technical Report WC2A 2AE, London School of Economics and Political Science.

[Ben-Or and Linial 1989] Michael Ben-Or and Nathan Linial. 1989. Collective coin flipping. *Randomness and Computation*, pages 91–125.

[Boros et al, 2003] Endre Boros, Toshihide Ibaraki, and Kazuhisa Makino. 2003. Variations on extending partially defined boolean functions with missing bits. *Information and Computation*, 180:53–70.

[Bshouty 1995] Nader H. Bshouty. 1995. Exact learning boolean functions via the monotone theory. *Information and Computation*, 123:146–153.

[Carreras and Freixas 1996] Francesc Carreras and Josep Freixas. 1996. Complete simple games. *Mathematical Social Sciences*, 32:139–155.

[Chow 1961a] C. K. Chow. 1961a. Boolean functions realizable with single threshold functions. *Proceedings of the IRE*, 49:370–371.

[Chow 1961b] C. K. Chow. 1961b. On the characterization of threshold functions. *Proceedings of the second Annual Symposium and Papers from the first Annual Symposium on Switching Circuit Theory and Logical Design*, S-134:34–38.

[Coates and Lewis 1961] C. L. Coates and P.M. Lewis. 1961. Linearly separable switching functions. *Franklin Institute Journal*, 272(5):366–410.

[Coleman 1961] Robert P. Coleman. 1961. Orthogonal functions for the logical design of switching circuits. *IRE Transactions on Electronic Computers*, EC-10:379–383.

[Dunne et al, 1995] Paul E. Dunne, Paul H. Leng, and Gerald F. Nwana. 1995. On the complexity of boolean functions computed by lazy oracles. *IEEE Transactions on Computers*, 44(4):495–502.

[Einy and Lehrer 1989] E. Einy and E. Lehrer. 1989. Regular simple games. *International Journal of Game Theory*, 18:195–207.

[Elgot 1961] C. C. Elgot. 1961. Truth functions realizable by a single threshold organ. *Switching Circuit and Logical Design*, pages 225–245.

[Eui-Hong et al, 1997] Han Eui-Hong, George Karypis, Vipin Kumar, and Banshad Mobasher. 1997. Clustering in a high-dimensional space using hypergraph models. Technical Report TR-97-049, Army HPC Research Center, University of Minnesota.

[Gallant 1990] S.I. Gallant. 1990. Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 1(2):179–191.

[Garey and Johnson 1979] Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Bell Telephone Laboratories, New Jersey.

[Golomb 1959] Solomon W. Golomb. 1959. On classification of boolean functions. *IRE Transactions on Information Theory*, 5(5):176–186.

[Golubchick and Lui 2001] Leana Golubchick and John C.S. Lui. 2001. Open problems for threshold-based systems. *ACM SIGMETRICS Performance Evaluation*

Review, 28(4):6–8.

[Goodman and O'Rourke 2004] Jacob E. Goodman and Joseph O'Rourke, editors. 2004. *Handbook of Discrete and Computational Geometry.* Chapman & Hall.

[Goto and Takahasi 1963] E. Goto and H. Takahasi. 1963. Some theorems useful in threshold logic for enumerating boolean functions. *Proceedings of IFIP Congress 62*, pages 747–752.

[Grünbaum 2003] Branko Grünbaum. 2003. *Convex Polytopes.* Springer-Verlag New York, Inc., New York, NY.

[Hagan et al, 1996] Martin Hagan, Howard Demuth, and Mark Beale. 1996. *Neural Network Design.* PWS Publishing Co., Boston, MA.

[Hammer et al, 1981] P.L. Hammer, T. Ibaraki, and U.N. Peled. 1981. Threshold numbers and threshold completions. *Annuls of Discrete Mathematics*, 11:125–145.

[Hammer et al, 2000] Peter L. Hammer, Alexander Kogan, and Uriel G. Rothblum. 2000. Evaluation, strength and relevance of variables of boolean functions. *SIAM Journal of Discrete Math*, 13(3):302–312.

[Håsted 1994] Johan Håsted. 1994. On the size of weights for threshold gates. *SIAM Journal of Discrete Math*, 7(3):483–492.

[Hegedüs and Megiddo 1996] Tibor Hegedüs and Nimrod Megiddo. 1996. On the ge-

ometric separability of boolean functions. *Discrete Applied Mathematics*, 66:205–218.

[Horváth 1994] E.K. Horváth. 1994. Invariance groups of threshold functions. *Acta Cybernetica.*

[Hu 1965] Sze-Tsen Hu. 1965. *Threshold Logic.* University of California Press, Berkeley.

[Irmatov 1996] A. A. Irmatov. 1996. Bounds for the number of threshold functions. *Discrete Mathematics and Applications*, 6(6):569–583.

[Kahn et al, 1998] Jeff Kahn, Gil Kalai, and Nathan Linial. 1998. The influence of variables on boolean functions. *Foundations of Computer Science, 29th Annual Symposium on*, pages 68–80.

[Kaszerman 1963] Philip Kaszerman. 1963. A geometric test-synthesis procedure for a threshold device. *Information and Control*, 6:381–398.

[Keener 2004] Lee L. Keener. 2004. Linear separation of the vertices of a hypercube. *Mathematics and Physics Symposium, UNBC.*

[Lazarte 1989] Emamy Lazarte. 1989. On the cut complexes of the 5-cube. *Discrete Mathematics*, 78:239–256.

[Littlestone 1988] Nick Littlestone. 1988. Learning quickly when irrelevent at-

tributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318.

[Muroga et al, 1962] S. Muroga, I. Toda, and M. Kondo. 1962. Majoriy decision functions of up to six variables. *Mathematics of Computation*, 16:459–472.

[Muroga et al, 1970] Muroga, Tsuboi, and Baugh. 1970. Enumeration of threshold functions of eight variables. *IEEE Transactions on Computers*, C-19(9):818–825.

[Muroga 1965] Saburo Muroga. 1965. Lower bounds on the number of threshold functions and a maximum weight. *IEEE Trans. on Electronic Computers*, EC-14:136–148.

[Muroga 1971] Saburo Muroga. 1971. *Threshold Logic and Its Applications.* John Wiley & Sons, New York, NY.

[Newman and Wigderson 2002] Ilan Newman and Avi Wigderson. 2002. Lower bounds on formula size of boolean functions using hypergraph-entropy. *SIAM Journal on Discrete Mathematics*, 8(4):536–542.

[Ojha 2000] P.C. Ojha. 2000. Enumeration of linear threshold functions from the lattice of hyperplane intersections. *EEE Transactions on Neural Networks*, 11(4):839–850.

[Paull and McCluskey 1960] M.C. Paull and E.J. McCluskey. 1960. Boolean function realizeable with single threshold devices. *Proceedings of the IRE*, 48:1335–1337.

[Picton 2000] Phil Picton. 2000. *Neural Networks.* Palgrave, New York, NY, second edition.

[Picton 2001] P.D. Picton. 2001. Deriving weights for single threshold logic gates using decomposition. Technical report, School of Technology and Design, University College Northampton.

[Quintana et al, 1997] Jose Quintana, Maria Avedillo, Raul Jimenez, and Ester Rodriguez-Villegas. 1997. Practical low-cost cpl implementation of threshold logic functions. *Proc. Great Lakes Symp. VLSI*, pages 139–144.

[Ramos et al, 2003] Jose Ramos, Javier Garcia, Santiago Martin, and Alfonso Bohorquez. 2003. A new theorem in threshold logic and its application to multioperand binary adders. *International Journal of Computer Mathematics*, 80(11):1363–1372.

[Saks 1993] Michael E. Saks. 1993. Slicing the hypercube. *Surveys in Combinatorics*, 187:211–255.

[Schläfli 1850] Ludwig Schläfli. 1850. Gesammelte mathematische abhandlugen. *Band*, 1.

[Schnelle and Engel 1991] T. Schnelle and A. Engel. 1991. Robustness and information capacity of learning rules for neural network models. *Physics Letters A*, 156(1-2):69–75.

[Sima and Orponen 2003] Jiri Sima and Pekka Orponen. 2003. General-purpose

computation with neural netwroks: A survey of complexity theoretic results. *Neural Computation*, 15:2727–2778.

[Slepian 1953] David Slepian. 1953. On the number of symmetry types of boolean functions of n variables. *Canadian Journal of Mathematics*, V(2):185–193.

[Taylor and Zwicker 1995] Alan Taylor and William Zwicker. 1995. Simple games and magic squares. *Journal of Combinatorial Theory, Series A*, 71:67–88.

[Taylor and Zwicker 1999] Alan D. Taylor and William S. Zwicker. 1999. *Simple Games : Desireability Relations, Trading, Pseudoweightings.* Princeton University Press, Princeton, New Jersey.

[Wegener 1987] Ingo Wegener. 1987. *The Complexity of Boolean Functions.* John Wiley & Sons, Chichester.

[Winder 1961] R. O. Winder. 1961. Single stage threshold logic. *Switching Circuit and Logical Design*, S-134:321–332.

[Winder 1965] R. O. Winder. 1965. Enumeration of seven-argument threshold functions. *IEEE Transactions on Electronic Computers*, 14(3):315–325.

[Winder 1968] Robert O. Winder. 1968. Symmetry types in threshold logic. *IEEE Transactions on Computers*, 17(1):75–78.

[Winder 1969] R. O. Winder. 1969. Threshold gate approximations based on chow

parameters. *IEEE Transactions on Computers*, C18(4):372–375.

[Winder 1971] Robert O. Winder. 1971. Chow parameters in threshold logic. *Journal of the Association for Computing Machinery*, 18(2):265–289.

[Yamija and Ibaraki 1965] S. Yamija and T. Ibaraki. 1965. A lower bound of the number of threshold functions. *IEEE Transactions on Electronic Computers*, 14:926–929.

[Zuev and Lipkin 1988] Y. A. Zuev and L. I. Lipkin. 1988. Estimating the efficiency of threshold representations of threshold functions. *Cybernetics*, 24:713–723.

[Zuev 1989] Y. A. Zuev. 1989. Asymptotics of the logarithm of the number of threshold functions of the algebra of logic. *Soviet Mathematics Doklady*, 39:512–513.