THE COMPLEXITY OF MONOTONE BOOLEAN FUNCTIONS AND AN ALGORITHM FOR FINDING SHORTEST PATHS IN A GRAPH

by

Peter Anthony Bloniarz

January 1979

This report is a minor revision of a thesis submitted in August, 1977 in partial satisfaction of the requirements for the degree of Doctor of Philosophy in Computer Science

This research was partially supported by National Science Foundation Grants MCS76-14294, MCS74-12997-A04, and MCS77-19754. The writing was partially supported by NSF Grant MCS78-04346.

Massachusetts Institute of Technology
Laboratory for Computer Science

Cambridge Massachusetts 02139

THE COMPLEXITY OF MONOTONE BOOLEAN FUNCTIONS AND AN ALGORITHM FOR FINDING SHORTEST PATHS IN A GRAPH

by

PETER ANTHONY BLONIARZ

Submitted to the

Department of Electrical Engineering and Computer Science
on August 12, 1977 in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy

ABSTRACT

The first part of this thesis considers the complexity of Boolean functions. The main complexity measures used are the number of gates in combinational networks and the size of Boolean formulas. The case of monotone realizations, using only the operations of AND and OR, of monotone functions is emphasized.

For a particular class of monotone functions, the quadratic functions, the worst-case values for the monotone circuit complexity is shown to be proportional to $n^2/\log n$. The number of $\wedge$-gates necessary to compute any quadrastic function is also analyzed.

A technique for deriving bounds on monotone circuit size of threshold functions is applied to the "majority" function (threshold $n/2$) to establish a lower bound on its monotone circuit complexity of $3n-O(1)$. For the function "threshold 2", previously known lower bounds on the number of $\vee$-gates required are extended in the case of a circuit which has a minimal number of $\wedge$-gates.

As a result, it follows that no monotone circuit for "threshold 2" can simultaneously have both a minimal number of $\wedge$-gates and a minimal number of $\vee$-gates.

The complexity of combinations of functions on disjoint sets of variables is studied, and a gap between the formula and circuit size of a particular function is given.

Finally, we study the effect of allowing negation in a formula for monotonic functions. Examples are given both of functions in which using negations allows more succinct expressions, and functions in which it does not.

The second part of the thesis describes an algorithm for computing shortest paths in a graph. These results show that an algorithm originally proposed by Spira for this problem can have slow running time. The lacuna in his algorithm is repaired, and it is shown to have $O(n^2(\log n)^2)$ average running time over wide classes of graphs as Spira originally claimed. As a special case, a transitive closure algorithm with $O(n^2 \log n)$ average time is also described.

Thesis Supervisor: Albert R. Meyer, Professor of Computer Science.

Key Words: Boolean functions, circuit complexity, Boolean formula size, monotone networks, threshold functions, quadratic functions, algorithms, shortest paths, directed graphs, transitive closure, computational complexity.

ACKNOWLEDGEMENTS

This thesis is dedicated to my friend and companion Mary Ellen. Thanks for everything.

I'd like to thank all my friends who made my life more enjoyable because I knew them. Thanks to Albert and Ron for showing me what research was all about, and for having the faith and patience to see me through this. Thanks to Paul and Mary for their own unique contributions to the completion of this manuscript. Thanks to Wendy for typing this during her vacation, and Peter Elias for struggling through many versions of this thesis.

TABLE OF CONTENTS

CHAPTER 1

Introduction

This thesis consists of two parts, each devoted to a different area in the theory of computation. It is primarily devoted to a study of the complexity of Boolean functions, with an emphasis on monotone Boolean functions. A secondary objective is the analysis of algorithms which compute the shortest distances between all points in a graph and which compute the transitive closure of a Boolean matrix.

In general, computational complexity theory asks questions about the computational resources required to solve a problem or compute a function. Traditional complexity theory has been primarily concerned with the difficulty of computing recursive functions on various models of machines such as Turing Machines or Random Access Machines; measures such as time or space on these machines are considered. Finite functions (i.e. those with finite domain), since they are computable by finite state machines, are inherently "easy" according to these definitions since they require only enough time and space to read the input and print the output. Several models of computing machines for finite functions have been proposed over the past years, and these have been studied as a means for defining the complexity of finite functions. Some of these, such as circuit complexity, receive their motivation from computer technology, while others such as formula size are more mathematical in origin. In the work which has been done so far, these models are seen to have a rich mathematical structure which is only now beginning to be understood to any extent. The bulk of the research contained in this thesis is a study of several problems in this area of finite computation.

We restrict ourselves to the Boolean case in which the finite domain and range are vector spaces over the set {0,1}. The original work in this area was directed towards finding asymptotic results about the worst-case complexity of all Boolean functions on n variables. Work of Shannon [1949], Lupanov [1958,1962], and others [Krichevskii 1961] established that "most" Boolean functions on n variables have minimal formulas of size asymptotic with $2^n/\log n$,† and minimal circuits of size asymptotic with $2^n/n$. During the 1950's attention was focused more on individual "practical" problems, and on finding optimal or near-optimal canonical formulas and circuits for a function with a given input-output specification. Work of Quine [1952,1955] and others [McCluskey 1956, Karnaugh 1953] explored methods for finding efficient ways of computing or expressing a specific function.

Recent research has been directed primarily at finding exact values for the complexity of individual functions in different models of complexity. As an adjunct to this pursuit, other more

† In this report, all logarithms are to the base 2.

basic questions such as the relationship between complexity in different models have been studied. This search for optimal circuits and formulas is only indirectly motivated from an engineering point of view[†], but is relevant to complexity theory.

Several authors [Pippenger and Fischer 1977, Schnorr 1976d] have demonstrated a connection between the Turing Machine complexity of a function and the circuit complexity of finite restrictions of that function. These results essentially show that a function is easy to compute with respect to oracle Turing Machine time if and only if it has a small circuit on each set of finite length inputs. This work relates to recent studies of non-deterministic polynomial time (*NP*) and to the *NP*-complete problems of Cook [1971] and Karp [1972] because it implies that if $P = NP$, then the finite restriction of any problem in *NP* has a polynomial-sized circuit. While this latter question is obviously still open, this observation has given impetus to many researchers to explore the possibility of proving large lower bounds on the complexity of specific Boolean functions.

[†] Considerations such as the number of gates in a circuit or the delay time are not as important as other factors such as the interconnections between gates and the fan-out of the gates.

The results obtained thus far have not been very encouraging in pursuit of this goal. The lower bounds obtained may be divided into two types. Diagonal-type arguments can obtain exponential lower bounds on the complexity of specific functions. For example, Stockmeyer [1974] and Meyer have shown that, since there are functions which are recognizable on a deterministic Turing Machine in space 2^n whose restrictions to inputs of fixed length have exponential circuit complexity, one can get a lower bound of c^n on any circuit which computes a restriction of certain logical languages. Thus far, results of this sort have been limited to Boolean functions which are binary codings of problems which are provably difficult to compute on a Turing Machine; the functions involved have not arisen as Boolean functions in their own right. Moreover, since the *NP*-complete problems are not yet provably difficult for a Turing Machine, these techniques do not apply to them.

Direct arguments for particular functions have not yielded much in the way of lower bounds. For single-output functions, no lower bounds on circuit complexity larger than linear in the number of inputs have yet been proven.

The order of these bounds has not been improved even when the allowable gates in the circuit are highly restricted. For the complete basis of all binary Boolean operations, the largest of these lower bounds is 2.5n recently discovered by Paul [1977] and Stockmeyer [1977]. [†]For restricted bases, the largest known lower bound is 7n for particular functions [Redkin 1973]. Work contained in this thesis establishes lower bounds on the monotone circuit complexity of a particular threshold function which is equal to the largest lower bound achieved for this basis (namely 3n).

For multi-output functions, larger lower bounds have been achieved on the circuit complexity in certain bases by exploiting the interplay between the functions computed by different outputs. For example, Paterson [1975] and Mehlhorn and Galil [1976], indepently extending earlier work of Pratt [1974], have shown that the monotone circuit complexity of the multiplication of two nxn matrices is $2n^3-n^2$. [††]Several other researchers have used different techniques to obtain lower bounds proportional to n log n on functions related to sorting problems [Lamagna and Savage 1974, Lamagna 1975].

We observe that in both cases the bounds obtained are far short of the exponential lower bounds theoretically possible for most functions.

[†] Recently, Lipton and Tarjon [1977] exhibited larger bounds for planar circuits, and Schnorr [1978b] has announced a 3n lower bound for arbitrary circuits.

[††] Recently, these techniques have been extended to yield an $\Omega(n^2/\log^2 n)$ lower bound on the monotone circuit complexity of certain functions [Wegener 1978]

For formulas the results are also limited. If arbitrary connectives are allowed in the formula, the largest lower bounds are given by a general technique of Nečiporuk [1966] which he uses to establish lower bounds proportional to $n^2/\log n$ for a specific function. Harper and Savage use Nečiporuk's method to establish lower bounds on the complexity of the marriage problem [1972]. This technique is used later in this thesis to exhibit a monotone function for which negations don't allow a reduction in the formula size by more than a constant factor. Other lower bounds for formulas in which arbitrary binary connectives are allowed have been reported by several authors [Hansel 1964, Hodes and Specker 1968, Vilfan 1972, and Fischer, Meyer and Paterson 1975]. For more restricted sets of connectives, Khrapchenko [1971] established lower bounds on the $\{\wedge,\vee,\neg\}$ formula complexity for the parity function proportional to n^2, and this is the largest polynomial bound yet established for formula complexity.

The principal research contained in this thesis is the establishment of lower bounds on the complexity of specific Boolean functions, but includes more general work on asymptotic bounds on the complexity classes of Boolean functions and on the relationship between several complexity measures. In particular, we begin by studying the monotone circuit complexity of quadratic monotone Boolean functions. (Those for which each prime implicant is a product of two distinct variables.) Using simple counting techniques we show that most quadratic functions require at least $\Omega(n^2/\log n)$[†] gates to compute in any circuit. On the other hand, we show that a monotone circuit of size proportional to $n^2/\log n$ exists for every quadratic function. When counting individual types of gates in a monotone circuit, we show that $n-1$ $\wedge$-gates suffice to compute any quadratic function, and exhibit a specific function for which $2n/3$ $\wedge$-gates are necessary in any monotone circuit.

In chapter 4 we examine the question of whether it may be easier to compute combinations of several functions than to compute the individual functions and then combine them. Both monotone and general circuits and formulas are considered. As a corollary to the work reported there, we establish the existence of a function of polynomial complexity which has smaller circuit complexity than formula complexity -- in fact their ratio is proportional to $n/\log \log n$.

Chapter 5 contains an examination of the monotone circuit complexity of the threshold functions. Using techniques similar to Paul [1977], Stockmeyer [1977], and Schnorr [1974], we establish larger lower bounds on their complexity than previously known. These bounds are in some cases quite small in comparison with the best known upper bounds. In particular, for the function "threshold n/2" of n variables, we establish a lower bound of

[†] For notation, see page 51.

3n-7 gates necessary in any monotone circuit; [†]on the other hand, the smallest known circuit has a number of gates proportional to $n(\log n)^2$. For the function "threshold 2" of n variables, we present exact bounds on the number of $\wedge$-gates and $\vee$-gates for any monotone circuit. These bounds were observed by F.F. Yao[††]. We extend these bounds in the case of a circuit with the minimal number of $\wedge$-gates for values of n which are a power of 2, and show that for these values of n it is impossible to simultaneously achieve the minimum number of both types of gates. The latter result was again previously observed by F.F. Yao.

Finally, we study the effect of different bases on the formula complexity of monotone functions. In work done jointly with M. Paterson, we exhibit a function with monotone formula complexity proportional to $n^2/\log n$ for which any formula in an arbitrary basis can be no more than a constant factor smaller. On the other hand, in work done jointly with A. Meyer, we show that there are monotone Boolean functions for which the smallest monotone formula is larger by a factor of $\Theta(n)$ than a formula for the function in the basis of all binary operators.

[†] Recently these techniques have been extended by the author to yield a 3n-O(1) lower bound on the $\{\wedge, \vee, \neg\}$ complexity of this function.

[††] Personal communication, 1975.

The final part of this thesis deals with the problems of determining the transitive closure of a Boolean matrix and determining the shortest distances between all points of a non-negatively weighted graph. P.M. Spira [1973] has published an algorithm to find the shortest distance matrix which he claims has an average running time (over a large class of weighted directed graphs) of $O(n^2 \log^2 n)$. Research by A. Meyer, M.J. Fischer, and this author has demonstrated classes for which his algorithm has slow average running time. A revision of his algorithm does indeed have $O(n^2 \log^2 n)$ average time, over even wider classes of graphs than Spira claimed. This result is presented in Chapter 7; it is also shown there that a further revision yields a simple $O(n^2 \log n)$ average time algorithm to compute the Boolean transitive closure over a wide class of probability distributions on matrices.

We begin by presenting some introductory remarks on Boolean functions.

CHAPTER 2

Definitions and Preliminary Results

Section A. Boolean Functions

An n-input, m-output Boolean function is a function with domain $\{0,1\}^n$ and range $\{0,1\}^m$. We will denote the set of all n-input, m-output Boolean functions by $B_{n,m}$. We will generally be interested in one-output Boolean functions and denote the set $B_{n,1}$ of all such functions with n inputs as B_n. If $f \in B_{n,m}$, we usually denote an arbitrary element of the domain of f by $\vec{x} = (x_1, x_2, \ldots, x_n)$, and refer to each x_i for $1 \leq i \leq n$ as a variable of f. Some specific Boolean functions we will refer to are the unary function negation or NOT (denoted $\neg$), and the binary functions disjunction or OR ($\vee$), conjunction or AND ($\wedge$ or $\cdot$)†, EXCLUSIVE OR or mod 2 sum ($\oplus$), equivalence ($\equiv$); and the constant functions 0 and 1.

The sets of functions $B_{n,m}$ have been extensively studied from a mathematical point of view [Harrison 1965, MacLane and Birkhoff 1967]. We assume that the reader is familiar with elementary properties of these sets. We use notation which is standard and refer the reader to Harrison [1965] for elaboration and proofs of

† Frequently we will denote the conjunction of variables x and y by their concatenation xy.

results that we only state[†].

Boolean functions may be combined in standard fashion. In particular the composition of functions is defined as usual. If f and g are members of B_n for some n, we define $f \vee g \varepsilon B_n$ by $(f \vee g)(\vec{x}) = f(\vec{x}) \vee g(\vec{x})$ for any $\vec{x} \varepsilon \{0,1\}^n$. If a collection of functions $(f_i | i \varepsilon I)$ is each a member of B_n, the disjunction $\bigvee_{i \varepsilon I} f_i \varepsilon B_n$ is defined in the obvious fashion. The conjunction of functions is defined similarly. In addition, we may define, from the functions $f \varepsilon B_{n_1,m_1}$ and $g \varepsilon B_{n_2,m_2}$, the function $f x g \varepsilon B_{n_1+n_2,m_1+m_2}$ by

$$fxg(x_1,x_2,\ldots,x_{n_1},y_1,y_2,\ldots,y_{n_2}) = (f(x_1,\ldots,x_{n_1}), g(y_1,\ldots,y_{n_2}))$$

(where we use the obvious isomorphism between $\{0,1\}^{m_1} x \{0,1\}^{m_2}$ and $\{0,1\}^{m_1+m_2}$). The combination $v(fxg)$ is defined, for $f \varepsilon B_{n_1}$

[†] Examination of the properties of Boolean functions indicates strong similarities among them. In fact, they may be grouped into pairs in which occurrences of $\wedge$ and $\vee$ are interchanged, as well as occurrences of 0 and 1. Such statements are called duals of each other. It is a well known fact that a Boolean statement is true if and only if its dual is true. This is known as the pricipal of duality, and permits some economy in proving properties of Boolean functions.

and $g \epsilon B_{n_2}$, to be the obvious composition of $\vee$ and $\times$; that is $\vee(f\times g)(x_1,\ldots,x_{n_1},y_1,\ldots,y_{n_2}) = f(x_1,\ldots,x_{n_1}) \vee g(y_1,\ldots,y_{n_2})$. $\wedge(f\times g)$ is defined similarly.

Two properties of these combinations which we will use are stated in the following lemma:

2.1 Lemma: Suppose f and g are members of B_n for some $n \geq 0$. Then (a) if $f \vee g = \underline{0}$, the constant function 0, then $f = 0$ and $g = 0$

and (b) if $f \wedge g = \underline{1}$, then $f = \underline{1}$ and $g = \underline{1}$.

One specific set of Boolean functions is the set of projection functions. The i^{th} projection function of n variables, denoted Π_i^n, is defined by $\Pi_i^n(x_1,\ldots,x_n) = x_i$ for each i in $\{1,2,\ldots,n\}$. If f is an arbitrary member of $B_{n,m}$, we denote by f_i the function $\Pi_i^m \circ f$, the i^{th} component of f.

We say that a function $f \epsilon B_{n,m}$ functionally depends on its i^{th} variable x_i if there are constants $\vec{c}$ and $\vec{d}$ in $\{0,1\}^n$ which differ only in their i^{th} positions for which $f(\vec{c}) \neq f(\vec{d})$. Finally, if A is some set of input variables for f, and $c_x \epsilon \{0,1\}$ is a constant defined for each $x \epsilon A$, then we denote the restricted function of the remaining variables which is obtained from f by setting each variable x in A to c_x by $f\,|_{x = c_x \text{ for } x \epsilon A}$.

We are particularly interested in one subclass of Boolean functions, the monotone Boolean functions (m.b.f.'s). Observe that the set $\{0,1\}^n$ is partially ordered by the rule

$$(x_1,\ldots,x_n) \leq (y_1,\ldots,y_n) \text{ iff } x_i \leq y_i \text{ for } i = 1,2,\ldots,n$$

(where $0 \leq 1$).

The set $B_{n.m}$ is partially ordered (in fact is a Boolean algebra) by defining, for $f,g \in B_{n,m}$,

$$f \leq g \text{ iff } f(\vec{x}) \leq g(\vec{x}) \text{ for all } \vec{x} \in \{0,1\}^n.$$

A Boolean function $f \in B^{n,m}$ is <u>monotonic increasing</u>, or <u>monotone</u> for short, if and only if it preserves the partial ordering $\leq$; that is, if $f(\vec{x}) \leq f(\vec{y})$ for every pair of inputs $\vec{x},\vec{y} \in \{0,1\}^n$ for which $\vec{x} \leq \vec{y}$.

Specific examples of m.b.f.'s include the function <u>threshold k</u> of n variables, denoted T_k^n, which is defined by $T_k^n(x_1,\ldots,x_n) = 1$ if and only if at least k of the variables $(x_1,\ldots,x_n)$ have the value 1. The threshold 1 function is the disjunction $x_1 \vee x_2 \vee \ldots \vee x_n$; the threshold n function is the conjunction $x_1 \wedge x_2 \wedge \ldots \wedge x_n$. Another m.b.f. is the Boolean multiplication of two $n \times n$ matrices A and B, a function in B_{2n^2,n^2} which is defined by $A \cdot B = C$ where $C_{ij} = \bigvee_{k=1}^{n} (A_{ik} \wedge B_{kj})$ for each i and j in $\{1,2,\ldots,n\}$. A third set of m.b.f.'s is Boolean convolution, in which we define $f \in B_{n,2n-1}$ by

$$f_k(x_0,\ldots,x_{n-1},y_0,\ldots,y_{n-1}) = \bigvee_{i+j=k}(x_i \wedge y_j)$$

for each $0 \le k \le 2n-2$.

The m.b.f.'s satisfy several properties which are not true of Boolean functions in general. The following facts are easily verified.

2.2 Lemma: Suppose f and $g \epsilon B_n$ are monotone Boolean functions. Then

(a) if $f \wedge g = \underline{0}$, then $f = \underline{0}$ or $g = \underline{0}$

and (b) if $f \vee g = \underline{1}$, then $f = \underline{1}$ or $g = \underline{1}$.

Further properties of the m.b.f.'s will be explored later in this chapter. It is useful at this point to point out one property of the ordering relationship on B_n.

2.3 Lemma: Suppose that f, g, and h are functions in B_n. If $f \le h$ and $g \le h$, then

(1) $f \wedge h = f$

(2) $f \vee h = h$

and (3) $f \wedge g \le f \le f \vee g \le h$. †

† We remark that the dual of an arbitrary function $f \epsilon B_n$ is the function $D(f)$ in B_n defined by $D(f)(\vec{x}) = \neg f(\neg x_1, \neg x_2, \ldots, \neg x_n)$. The truth of a statement

(cont. on next page)

Section B. Representations of Boolean Functions

There are many different ways of representing Boolean functions, some of which give rise to the complexity measures discussed in this thesis. One standard representation is the truth table of the value of the function on different inputs. The unifying representation used in this thesis is that of a *combinational* (or gate-type) *switching circuit*, and is identical with those studied by numerous authors [Harrison 1965, Savage 1976, Paul 1977, Schnorr 1974 and others]. The underlying structure is a graph, which we now define.

A *directed graph* D consists of a pair of finite sets (V,E), where V is a set whose members are called *nodes* or *vertices*, and $E \subseteq V \times V$ is a set whose members are called *edges* or *arcs*. Another notation for V and E is $\mathrm{NODES}(D)$ and $\mathrm{EDGES}(D)$ respectively. If $(v,w) \in \mathrm{EDGES}(D)$ we say there is an edge from v to w in D. A *path* or *chain of length* k in D is a sequence of nodes $v_0, v_1, \ldots, v_k$ (for $k \geq 0$) of D such that there is an edge from v_i to v_{i+1} for every $i \in \{0,1,\ldots,k-1\}$. A graph D is *acyclic* if there is no path from any node A of D to itself other than

like Lemma 2.3 is unchanged if every function is replaced by its dual.

Observe that if $f \leq g$, then $D(f) \geq D(g)$, and that the dual of a monotone function is monotone.

the trivial path of length 0. For any node A of an acyclic graph D, we define the following subsets of nodes of D which pertain to A:

Succ(A,D) = {B∈NODES(D) | there is an edge from A to B} is the set of immediate successors of A,

Succ^*(A,D) = {B∈NODES(D) | there is a path from A to B in D},

Succ^+(A,D) = {B∈NODES(D) | there is an path of length $k \geq 1$ from A to B},

Pred(A,D) = {B∈NODES(D) | there is an edge from B to A} is the set of immediate predecessors of A,

Pred^*(A,D) = {B∈NODES(D) | there is a path from B to A in D},

and Pred^+(A,D) = {B∈NODES(D) | there is a path of length $k \geq 1$ from B to A}.

If W is some subset of nodes of D, we may extend the above devinitions to W and speak, for example, of the set of immediate successors of W, defined by $\text{Succ}(W,D) = \bigcup_{A \in W} \text{Succ}(A,D)$. Finally, the indegree or fan-in (respectively outdegree or fan-out) of a node A is the number of edges directed into (out of) A in D and is denoted indeg(A,D) (outdeg(A,D)). In general, we will omit mention of the graph D in the above notation when the graph is clear from context.

Suppose Ω is a finite set of primitive functions $g_i \epsilon B_{n_i}$ called the basis. A combinational circuit or network over the basis Ω, in short an Ω-circuit, is a directed acyclic graph N together with a labelling of the nodes and edges of N which is subject to the following constraints and definitions:

(1) For some positive integer n, at most n of the nodes with indegree zero are given distinct labels from the set $\{x_1, x_2, \ldots, x_n\}$; we refer to such nodes as input nodes and denote their set by INPUTS(N).

(2) There may be one node of indegree zero labelled with the symbol ZERO, and there may be one node of indegree zero labelled with the symbol ONE; such nodes are referred to as constant nodes.

(3) There are no other nodes of indegree zero other than those mentioned in (1) or (2).

and (4) Each node with indegree one or more is labelled with some member g_i of Ω. Each such node is a gate node, and their set is denoted GATES(N). Such a node G labelled with g_i must have indegree in N equal to the arity n_i of g_i; furthermore, each edge directed into G must be labelled with one of the integers $\{1,2,\ldots,n_i\}$ in such a way that every edge into G gets a different label.

An example of such a network is given in Figure 2.1, where the basis is the set $\{\wedge,\vee,\neg\}$. (For figures, the arcs in the graph are always directed downward. Arbitrary nodes are designated by triangles, constants or inputs by rectangles, and gates by circles).

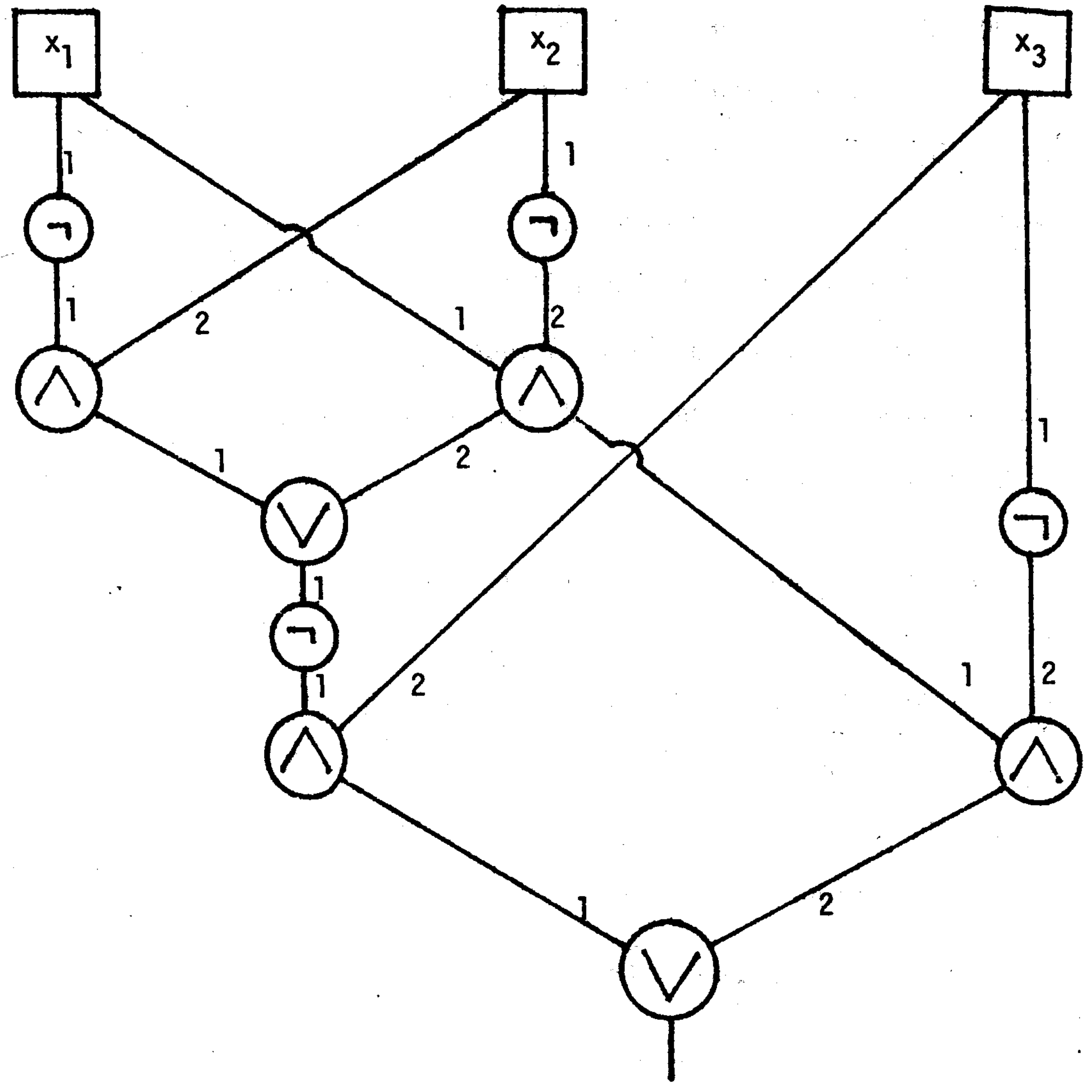


Fig. 2.1 A Combinational Circuit

If the input nodes to an Ω-circuit N are labelled with members of the set $\{x_1,\ldots,x_n\}$, we may associate a Boolean function in B_n with every node A in N by the following inductive rule. We denote the associated function by Res(A,N). If A is an input node labelled with the variable x_i, then Res(A,N) is the projection on the i^{th} coordinate, i.e. the function π_i^n. If A is the constant node ZERO (respectively ONE), then RES(A,N) = $\underline{0}$, the constant function in B_n equal to 0 on all inputs (respectively RES(A,N) = $\underline{1}$, the constant function in B_n equal to 1). Finally, suppose A is a gate for which every node in Pred(A) has an associated function. If we let B_k be the node in Pred(A) such that the edge from B_k to A is labelled with k, and if A is labelled with the basis element $g_i \in \Omega$, then

$$\mathrm{Res}(A,N) = g_i(\mathrm{Res}(B_1,N),\mathrm{Res}(B_2,N),\ldots,\mathrm{Res}(B_{n_i},N)).$$

We will say that an Ω-circuit N computes a function $f \in B_{n,m}$ iff for each i, $1 \leq i \leq m$, there is a node A_i of N such that $\mathrm{Res}(A_i,N) = f_i$, the i^{th} component of f. For example, the circuit given in Figure 2.1 computes the function $f(x_1,x_2,x_3) = x_1 \oplus x_2 \oplus x_3$ since the gate with outdegree 0 computes this function. We may also say that the circuit computes the function $g(x_1,x_2,x_3) = (x_1, x_1 \wedge \neg x_2)$ since there are gates which compute each component of g. A node which computes a component of the designated function will be called an output node.

A basis Ω is called semi-complete iff given any function $f \epsilon B_n$ for arbitrary n, there is an Ω-circuit as described above which computes f. The basis is called complete if any f can be computed by an Ω - circuit which does not contain the nodes ZERO and ONE (i.e. constants are not available). Three complete bases in which we will be interested are B_2, the set of all binary Boolean functions, $B = B_1 \cup B_2$, and $U = \{\wedge, \vee, \neg\}$. An example of a semi-complete basis is the set $\{\oplus, \wedge\}$. An exhaustive characterization of all complete bases has been made by Post [1941], and the interested reader is referred to this work.

One incomplete basis which holds special interest is the set $M = \{\wedge, \vee\}$ which is important for the following reason.

2.4 Theorem : A Boolean function $f \epsilon B_{n,m}$ is monotone iff there is an M-circuit which computes f.

For a proof of Theorem 2.4 the reader is referred to [Harrison 1965 p. 189].

The bulk of this thesis is concerned with the results about M-circuits (also called monotone circuits) computing monotone Boolean functions.

One special class of circuits which holds special interest is the set of Boolean formulas. An Ω-formula is an Ω-circuit in which each node in the circuit has outdegree at most one, and which has a unique node of fan-out zero corresponding to the unique output node. For a formula, we remove the restriction on the circuit that there be only a single node for each input variable and constant, and allow any number of nodes of indegree zero labelled with any input variable or constant, each node having outdegree one. For an example see Figure 2.2. The graph of any Ω-formula is a tree in which each leaf node (one with indegree zero) is labelled with a variable or constant, and each interior node (every other node) is labelled with a gate. This formulation is easily seen to be equivalent to the usual inductive definition of an Ω-formula. Clearly a formula can compute only a single-output function.

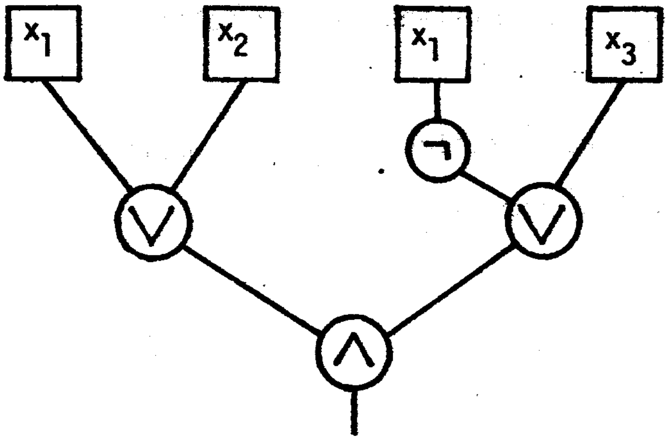


Fig. 2.2 The Formula $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3)$

Classically, Boolean formulas have been extensively studied as a means of representing Boolean functions. Many formula schemes have been proposed as canonical means for their representation, including the classical disjunctive normal form. In this method, a function $f \epsilon B_n$ is represented as the formula

$$\bigvee_{(c_1,\ldots,c_n)\epsilon A_f} x_1^{c_1} x_2^{c_2} \ldots x_n^{c_n}$$

where $$x^j = \begin{cases} x & \text{if } j = 1 \\ \neg x & \text{if } j = 0 \end{cases}$$

for any variable x, and A_f is those set of constants $\vec{c} \epsilon \{0,1\}^n$ for which $f(\vec{c}) = 1$. Other canonical forms are discussed in [Savage 1976].

If F is a formula, and H is a node in F, then F_H will denote the subformula of F above H; that is, $\mathrm{Pred}^*(H,F)$.

In this thesis, we will occasionally replace part of one circuit by another circuit and speak of combinations of circuits. For example, suppose that N and N' are Ω-circuits which contain nodes A and A' respectively. The <u>circuit obtained by replacing</u> <u>A by A'</u> is constructed by first identifying the corresponding input and constant nodes of N and N' and considering them as one network. Node A together with each arc directed into it is removed from the graph, and each edge (A,B) originally in circuit

N is replaced by the edge (A´,B) (with the same label as previously). For example, see Figure 2.3. The replacement of a node in a circuit N

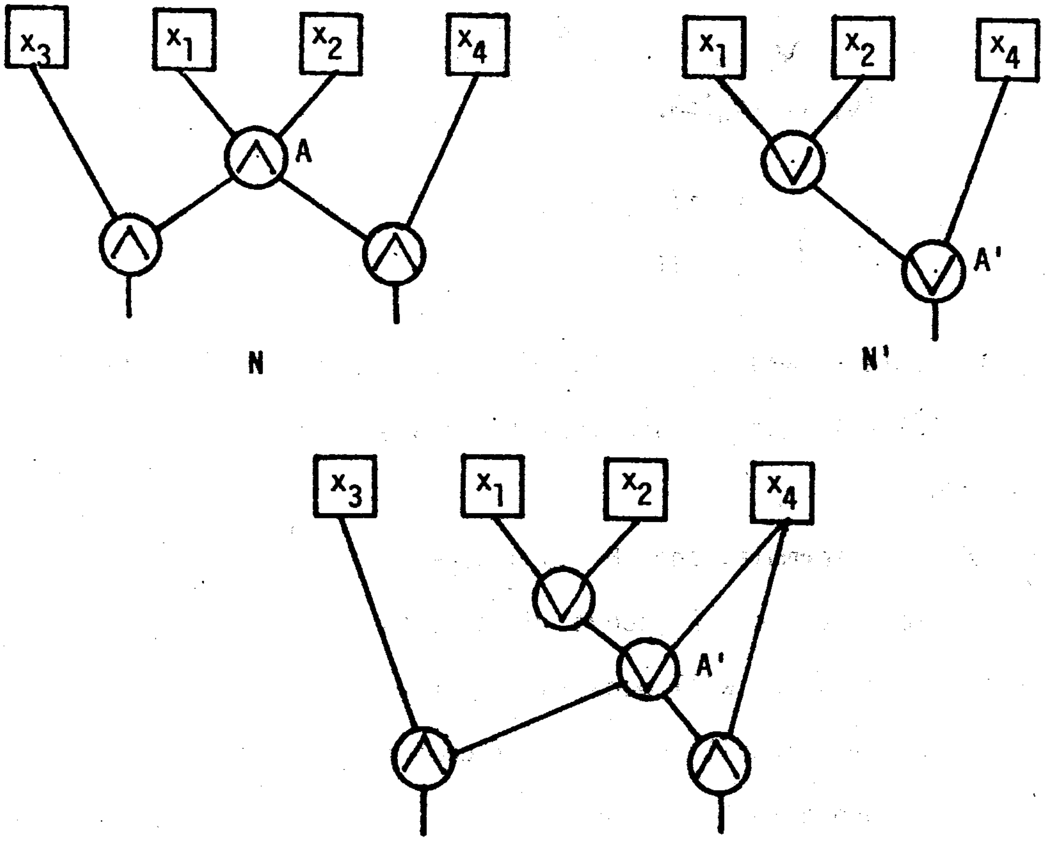


Fig. 2.3 The Replacement of A by A'

by a node in the same circuit is defined similarly, but substitution must not be made in a manner which introduces a cycle into the graph.

One may say little in general about the functions computed by a circuit in which one gate has been replaced by another. Specific situations may allow some conclusions to be drawn. For example, if node A in N is replaced by a node A′ in N′ for which Res(A,N) = Res(A′,N′), then it is easy to prove by induction on the length of paths that every remaining node in N computes the same function as it did before the substitution.

One special replacement is the substitution of a constant for a variable. We say that N′ is the circuit obtained from N _by setting variable_ $\underline{x_i}$ _to 0_ (respectively 1) if N′ is obtained by replacing variable node x_i in N by the constant node ZERO (respectively ONE). One inductively defines setting a collection of variables to a constant. It is again easy to verify the following fact.

2.5 Lemma: Suppose N is an Ω-circuit, and $x_i \in$ INPUTS(N). If N′ is the circuit obtained by setting variable x_i to 0 , then for each remaining node A in N′ ,

$$\mathrm{Res}(A,N') = \mathrm{Res}(A,N)|_{x_i = 0} \quad .$$

A similar statement may be made if x_1 is set to 1.

Observe that, depending on the basis, if a gate in a circuit has one input which is constant, then some simplification of the circuit may be made. For example, suppose the basis is the set B_2 of all binary Boolean functions. If a gate has a constant input then, except for trivial cases, it may be eliminated from the circuit since the output of that function is then a unary function of the other predecessor. Since this unary function may be absorbed into a preceding or succeding binary gate (if one such exists), then one may obtain a smaller circuit equivalent to the original. See Figure 2.4 for an example. Such simplifications may

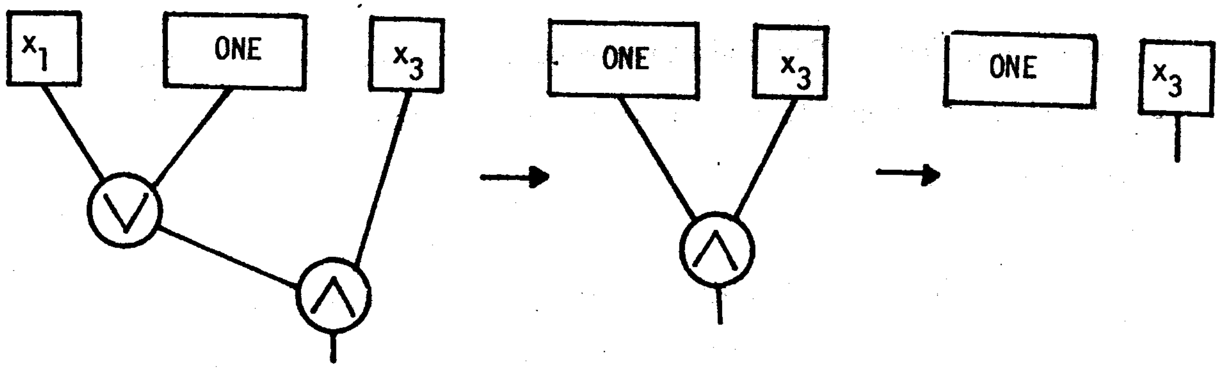


Fig. 2.4 Some Simplifications

always be made in the basis M. This elimination of gates will be a principal tool by which lower bounds on circuit complexity are obtained.

One additional representation for a Boolean function is as a set of prime implicants. In the interest of brevity we restrict ourselves to monotone Boolean functions, although a similar developmen may be made for all Boolean functions.

A *monome* or *product* is a product of distinct variables, e.g. $x_1x_2x_5$ is a monome. In particular, we denote the empty monome by ε. A monome m *includes* a monome m′ if every variable which appears in m′ also appears in m. We denote the function defined by a monome m by $T(m)$, and if M is a set of monomes, then $T(M)$ will denote the function $\bigvee_{m\in M} T(m)$. $T(\varepsilon)$ is the constant function $\underline{1}$.

Suppose that f is a monotone Boolean function. A monome m is an *implicant* of f iff $T(m) \leq f$; m is said to be a *prime implicant* of f if m includes no other implicant of f. We denote the set of all prime implicants of f by PI(f)[†].

For example, suppose $f(x_1,x_2,x_3)$ is the threshold 2 function of three variables, namely T_2^3 . Then $x_1x_2x_3$ is an implicant of f, but is not a prime implicant since the monome

[†] One defines $\bigvee_{m\in\emptyset} T(m) = \underline{0}$. Hence PI($\underline{0}$) = ∅, the empty set, and PI($\underline{1}$) = {ε}.

x_1x_2 is also an implicant of f. $PI(f)$ is the set $\{x_1x_2, x_2x_3, x_1x_3\}$.[†]

The importance of the set of prime implicants is that they are a canonical representation for the m.b.f.'s, and are fairly easy to manipulate mathematically. A straightforward proof (see [Harrison 1965 p. 190] for example) demonstrates the following fact of their canonicity.

<u>2.6 Lemma</u> (Quine): Suppose $f \in B_n$ is a monotone Boolean function. Then $f = \bigvee_{m \in PI(f)} T(m)$. Moreover, if $\mathbb{P}$ is any set of monomes such that no monome in $\mathbb{P}$ includes another monome in $\mathbb{P}$, and $f = T(\mathbb{P})$, then $\mathbb{P} = PI(f)$.

The set of prime implicants of a combination of functions can be obtained from those of the constituent functions by means of the following simple set of rules. Suppose that f and g are m.b.f.'s in B_n. Then for any monome m, m is an implicant of the function $f \vee g$ if and only if it is an implicant of f or an implicant of g.

[†] Note that we equate the monomes x_1x_2 and x_2x_1. Actually, the set of all monomes is equivalent to the free monoid on the elements $\{x_1, x_2, \ldots, x_n\}$ modulo the relations of the commutativity ($xy = yx$ for all variables x, y) and idempotency ($xx = x$ for all x). ϵ is the identity element for this algebraic system.

Moreover, $PI(f \vee g)$ is obtained from the union $PI(f) \cup PI(g)$ by eliminating all monomes which include another monome in the union. For example, if $PI(f_0) = \{x_1, x_2x_3, x_4\}$ and $PI(g_0) = \{x_1, x_2, x_3x_4\}$, then $PI(f_0 \vee g_0) = \{x_1, x_2, x_4\}$. If f and g are m.b.f.'s in B_n, and m is a monome such that $m \in PI(f \vee g)$, then $m \in PI(f)$ or $m \in PI(g)$, but the converse is not in general true. By Lemma 2.1(a), if $PI(f \vee g) = \emptyset$, then both $PI(f)$ and $PI(g)$ are empty sets.

The set of prime implicants of the function $f \wedge g$ may be obtained by forming the set of all products $m \cdot n$ of a monome $m \in PI(f)$ and a monome $n \in PI(g)$ (where duplicate variables in n and m are reduced by the rule $x \cdot x = x$), and then eliminating any products which include other products in the set. For f_0 and g_0 of the previous example, $PI(f_0 \wedge g_0) = \{x_1, x_2x_3, x_2x_4, x_3x_4\}$. Again, if $t \in PI(f \wedge g)$, then there are monomes $m \in PI(f)$ and $n \in PI(g)$ such that $m \cdot n = t$, but the converse is not in general true.

One may relate the ordering relation $\leq$ on the m.b.f.'s with their canonical representation using prime implicants by the following general result.

2.7 Lemma: Suppose f and $g \in B_n$ are m.b.f.'s. Then $f \leq g$ iff every monome $m \in PI(f)$ includes some monome $n \in PI(g)$.

Proof: Suppose $f \leq g$, and $m \in PI(f)$. By Lemma 2.6 and Lemma 2.3(c), $T(m) \leq \bigvee_{n \in PI(f)} T(n) = f \leq g$, so by the transitivity of $\leq$,

$T(m) \leq g$. Hence by definition m is an implicant of g. Define m_0 to be an implicant of g which is included in m and which includes no other implicant of g. Then m_0 is a prime implicant of g which is included in m.

Conversely, suppose every monome in PI(f) includes some monome in PI(g), and let $\vec{c}\epsilon\{0,1\}^n$ be such that $f(\vec{c}) = 1$. By Lemma 2.6, there is some monome $m \epsilon PI(f)$ such that $T(m)(\vec{c}) = 1$. By hypothesis, there is a monome n in PI(g) included in m ; By Lemma 2.3(c), we know that $T(m) \leq T(n)$. Hence $T(n)(\vec{c}) = 1$, and since n is an implicant of g, $g(\vec{c}) = 1$ as well.

□ Lemma 2.7.

One may define classes of Boolean functions in terms of structural properties of their set of prime implicants. For example, we will say that a m.b.f. $f\epsilon B_n$ is quadratic if every monome in PI(f) consists of the product of exactly two distinct variables; a m.b.f. $f\epsilon B_{n,m}$ is quadratic iff every component f_i is quadratic (for $1 \leq i \leq m$).

The set of prime implicants of a function has been extensively used as a tool in explaining arguments in this field [Paterson 1975 Mehlhorn and Galil 1976, Lamagna 1975]. This representation is particularly helpful in describing several replacements which may be made in M-circuits which don't affect the output function computed. Several general theorems have been elucidated by Mehlhorn and

Galil [1976] of which special cases have been used by other authors; we will find use for one of them later in the thesis. If N is any M-circuit, and A is a node in N, then we will denote $PI(Res(A,N))$ by $PI(A,N)$, and as usual omit mention of N whenever the circuit is obvious.

2.8 Lemma (Mehlhorn and Galil: Suppose N is an M-circuit, A is any node in N, and $PI(A) = \{m_0,m_1,\ldots,m_k\}$. If there is no output node B of N with $m_0 \cdot n \in PI(B)$ for some monome n, then one may replace A by a gate which computes the function $T(m_1,m_2,\ldots,m_k)$ without changing the function computed at any output gate.

For a proof of this result and several other general replacements for M-circuits the reader is referred to the paper [Mehlhorn and Galil 1976].

One final topic which pertains to circuits in general is the different notati‹ ›s of dependence. We will say that a node A in a circuit N depends functionally on the variable x_i if $Res(A,N)$ depends functionally on x_i. A different notion of structural (path or syntactic) dependence holds if there is a path from input node x_i to A in N, i.e. if $A \in Succ^*(x_i,N)$. Clearly if A depends functionally on x_i then it must also depend structurally on x_i, but in general the two notions do not coincide. It is not known whether they coincide for minimal sized circuits over particular bases (M for example). However, it is possible to prove the following useful observation.

2.9 Lemma: Suppose a function $f \epsilon B_n$ depends functionally on a variable x_i, and N is an Ω-circuit computing f with Res(U,N) = f for a node U in N. Then there is a path from x_i to U in N such that every gate in the path depends functionally on x_i.

Proof: We construct the chain backwards from the output gate U. Let $U = U_0$. Inductively assume that we have constructed a path $U_k, U_{k-1}, \ldots, U_0$ in N such that every node in the path depends on x_i. If $U_k = x_i$, then the proof is complete. Otherwise, U_k must be a gate since no other input or constant node depends on x_i. Hence, since $Res(U_k)$ depends on x_i, there must be a node $B \epsilon Pred(U_k)$ which depends functionally on x_i. We extend the path by defining $U_{k+1} = B$. (See Figure 2.5)

Since N contains no cycles, the nodes selected are all distinct. Since there are only a finite number of nodes in N, the above process must eventually halt.

□ Lemma 2.9

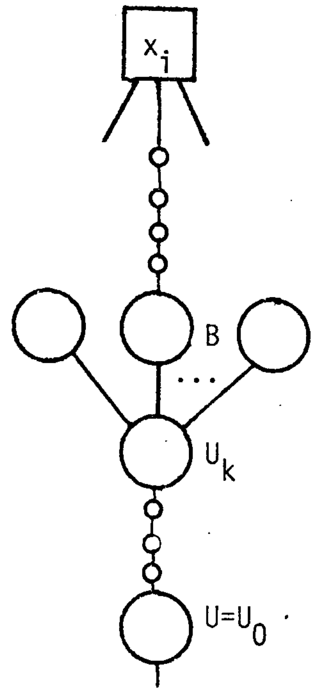


Fig. 2.5 The Extension

2.10 Corollary: If a function $f \epsilon B_n$, which depends on x_i, is not the projection function Π_i^n, and if N is an Ω-circuit which computes f, then outdeg(x_i,N) ≥ 1.

Section C. Complexity Measures

The models of computing Boolean functions introduced in the previous section naturally give rise to various complexity measures. If Ω is an arbitrary basis and N is an Ω-circuit, then we define $\hat{C}_{\Omega}(N)$ to be the number of gate-type nodes in N. If $f\epsilon B_{n,m}$, then we define the combinational (or circuit) complexity of f with respect to Ω, denoted $C_{\Omega}(f)$, to be the minimum value of $\hat{C}_{\Omega}(N)$ for all Ω-circuits N which compute f. If f is not definable by any Ω-circuit, we define $C_{\Omega}(f) = \infty$.

For formulas, one possible complexity measure is to consider the number of gates as was done above for circuits. We instead use the number of occurrences of literals as our complexity measure. If F is an Ω-formula, we define $\hat{L}_{\Omega}(F)$ to be the number of occurrences of variable input nodes in the circuit associated with F (the number of leaves of the tree corresponding to F which are labelled with variables). Observe that if Ω is a binary basis, and F contains no constant nodes, then $\hat{L}_{\Omega}(F)$ is one more than the number of gates in F. The formula complexity of a function $f\epsilon B_n$ with respect to Ω, denoted $L_{\Omega}(f)$, is the minimum of the values of $\hat{L}_{\Omega}(F)$ for all Ω-formulas F which define f. Again, if f is not definable by an Ω-formula, we define $L_{\Omega}(f) = \infty$. A minimal Ω-formula (respectively Ω-circuit) for f is an Ω-formula F (respectively Ω-circuit N) which computes f such that

$\hat{L}_\Omega(F) = L_\Omega(f)$ (respectively $\hat{C}_\Omega(N) = C_\Omega(f)$). Where there is no ambiguity we will use the same notations of complexity L_Ω for formulas and C_Ω for circuits as well as for functions.

For the particular monotone basis M, we will use the alternate notations MC and ML for C_M and L_M respectively. For this basis, we also define a complexity measure which depends on the types of gates used in the circuit or formula. If N is an M-circuit, define $MC_\vee(N)$ and $MC_\wedge(N)$ to be the number of $\vee$-gates and $\wedge$-gates respectively in N. If f is a m.b.f., we define $MC_\vee(f)$ (respectively $ML_\vee(f)$) to be the minimum value of $MC_\vee(N)$ as N ranges over all M-circuits (respectively formulas) which compute f (where here we consider a formula as a special type of circuit). We similarly define $MC_\wedge(f)$ and $ML_\wedge(f)$. An <u>$\vee$-minimal</u> (respectively <u>$\wedge$-minimal</u>) M-circuit for a m.b.f. f is one which has the minimal number $MC_\vee(f)$ of $\vee$-gates (respectively $MC_\wedge(f)$ of $\wedge$-gates).

The above are the complexity measures considered in the thesis, but other measures can be defined. For example, the depth of a circuit can be defined to be the length of the longest path in the circuit. This measure has been studied by Spira [1971] and more recently has been studied by several authors [McColl 1977, McColl and Paterson 1977, Paterson and Valiant 1976, and Borodin 1977]. One may similarly define the breadth of a circuit; this measure has recently been studied by Schnorr [1976b]. Finally, one may choose to consider only circuits in which the outdegree of of each gate in the circuit is bounded above by some integer k. Johnson, Savage, and Welch established the result that if $k \geq 2$, such a complexity measure is proportional to C_Ω for any complete basis Ω, (for proof, see [Savage 1976 p.24]).

The relationship between these complexity measures of an individual function is well understood in some circumstances and unknown in others. Muller [1956] pointed out that in general, the circuit complexity of a function in any complete basis Ω is proportional to its C_{B_2} complexity. This is true since any gate in B_2 may be synthesized by an Ω-subcircuit and vice versa, so one can replace a gate by a subcircuit with only a constant factor increase in the total number of gates[†]. For formulas, the choice of complete basis can make a difference in formula size. For the parity function $f_0(x_1,\ldots,x_n) = x_1 \oplus x_2 \oplus \ldots \oplus x_n$, Khrapchenko [1971] demonstrated that $L_U(f_0)$ is proportional to n^2, while f_0 clearly has a formula of size n in the basis B_2. Pratt [1975] has recently shown that the maximal gap between the complexity in any pair of binary complete bases cannot be much larger by demonstrating that $L_U(f) \leq [L_{B_2}(f)]^{\log_3 10}$ for any $f \in B_n$.

The relationship between the B_2-complexity of a monotone function and its complexity in the incomplete basis M is not as

[†] In fact, $C_B(f) = C_{B_2}(f)$ for any function $f \in B_n$, and if f depends on more than one variable, then $C_B(f)$ is the minimal number of binary gates in any B-circuit which computes f. See [Savage 1976 p. 39] for a proof of this fact.

well understood. Specific results will be shown in Chapter 6.

Results about the maximum values of these complexity measures among all Boolean functions were mentioned in the introduction. In brief, the number of distinct circuits with at most a fixed mumber k of gates is bounded above by $ck^{c'k}$ for some constants c and c' depending on the basis. From this fact, one can easily show that if there are many different functions to compute, then there are not enough small circuits to compute them all, so "most" of them require a large number of gates. What is remarkable is that soem clever constructions exist which enable one to compute all Boolean functions in certain classes with circuits or formulas which are close to, or even asymptotic with, the lower bounds on size obtained by the above argument.

For the class of all Boolean functions, Shannon [1949] and Lupanov [1958] have shown that the maximal B_2 circuit complexity of any Boolean function $f \in B_n$ is asymptotically $2^n/n$. Similarly, it is known [Krichevskii 1961, Lupanov 1962] that the maximal B_2 formula complexity is proportional to $2^n/\log n$.

Recent attention has been focused on the class of monotone Boolean functions. Kleitman and Markowsky [1975], extending work of numerous authors, have shown that there are asymptotically $2^{\binom{n}{\lfloor n/2 \rfloor}}$ m.b.f.'s of n variables, and one obtains a corresponding lower bound of order $2^n/n^{3/2}$ on the maximal B_2 (and M)

circuit complexity of m.b.f.'s. Pippenger [1976] has shown that this lower bound is attainable in showing that any m.b.f. $f \epsilon B_n$ can be computed in a B_2-circuit with asymptotically $O(2^n/n^{3/2})$ gates and an M-circuit with asymptotically $O(2^n \log n/n^{3/2})$ gates.

Lower bounds on the complexity of specific functions dominate the bulk of this thesis. It will be helpful to point out some facts about minimal circuits. If $f \epsilon B_{n,m}$ and Ω are arbitrary, then any minimal Ω-circuit for f has at most m gates of outdegree 0. This is so since any gate of outdegree 0 which is not an output gate may be eliminated from the circuit without changing the functions computed at the output nodes.

In a similar fashion, one can show that no two gates in a minimal Ω-circuit compute the same function. Suppose G and G' were distinct gates in a minimal Ω-circuit N for which $Res(G,N) = Res(G',N)$. Since N is acyclic, either $G \not\epsilon Succ^*(G')$ or $G' \not\epsilon Succ^*(G)$; w.l.o.g. we assume the former. Then we eliminate gate G, and replace G by G' in $Pred(H,N)$ for each gate $H \epsilon Succ(G,N)$. It is easily shown that the resulting circuit is acyclic, and that every remaining gate in the new circuit computes the same function as it did originally. This is a contradiction since N was a minimal Ω-circuit.

Suppose that a function $f \epsilon B_n$ depends on at least two variables Then any Ω-circuit or formula for f must contain at least one binary gate (where Ω is B_2, B, U, or M). By absorbing any unary gates into the binary gates, one can show that in any minimal Ω-circuit for f there are no edges directed out from any constant nodes since each gate with one input constant can be simplified and eliminated. Similarly, no node G has two edges directed into the same node H, so outdeg(G) is equal to the number of nodes in Succ(G).

For minimal formulas, we point out that the notions of structural and functional dependence on a variable x_i coincide, To prove this, suppose that Ω is an arbitrary basis, and F is a minimal Ω-formula for $f \in B_n$. If H is a gate in F which does not depend functionally on the variable x_i, then we claim that there are no occurrences of the variable x_i in F_H. If there were, then one could replace all occurrences of x_i in F_H by the constant 0 (or 1) without changing the function computed at H and hence by the entire formula F. Since this reduces the number of occurrences of inputs in F, F_H must not contain any occurrences of x_i.

Finally, we point out one simple lower bound on the complexity of most functions.

<u>2.11 Lemma</u> (Savage): Suppose a function $f \in B_n$ depends on each of its variables. Then $C_{B_2}(f) \geq n-1$.

Proof: Suppose N is a minimal B_2-circuit for f which has k gates. Observe that since each input node has outdegree at least one and since there is at most one gate of outdegree 0, we have

$$\sum_{A\epsilon \mathrm{Nodes}(N)} \mathrm{indegree}\ (A,N) = 2k$$

and

$$\sum_{A\epsilon \mathrm{Nodes}(N)} \mathrm{outdegree}(A,N) \geq n+k-1$$

Hence, since

$$\sum_{A\epsilon \mathrm{Nodes}(N)} \mathrm{indegree}(A,N) = \sum_{A\epsilon \mathrm{Nodes}(N)} \mathrm{outdegree}(A,N) ,$$

we obtain $2k \geq n+k-1$ which establishes the lemma.

□ Lemma 2.11

Section D. Circuits and Turing Machines

One question which naturally arises concerns the relationship between the measures of complexity defined above and ordinary Turing Machine (T.M.) complexity. As mentioned above, every finite function is "easy" for a Turing Machine to compute since it may be computed by finite state machine. When one considers the finite subproblems of an infinite problem, however, the situation is different. For concreteness, consider language recognition problems, i.e. functions $f:\{0,1\}^* \to \{0,1\}$ (where $\{0,1\}^*$ is the set of all finite-length words over the alphabet $\{0,1\}$), and let f_n be the restriction of f to words of length n. Consider the circuit complexity measure C_{B_2}.

One can show that there are functions f which are computable by a deterministic T.M. in space 2^n (i.e. a relatively easy function) for which $C_{B_2}(f_n)$ is near-maximal for all values of n (see [Stockmeyer 1974] for proof). On the other hand, one can exhibit arbitrarily complex, even arbitrarily non-recursive, functions f such that $C_{B_2}(f_n)$ is 1 for every $n \in \mathbb{N}$. There thus appears to be little correlation between the T.M. complexity of a function and the circuit complexity of its finite pieces.

A strong relationship between the two measures comes from a result due to Pippenger and Fischer [1977] which asserts the following: Suppose f is computable by a T.M. M operating in time $T(n)$ for all n. Then there is a constant c (depending on M's state diagram) such that $C(f_n) \leq cT(n)\log T(n)$. The proof is obtained by simulating machine M by an "oblivious" T.M. M' which operates in time $T \cdot \log T$ - the positions of the heads of M' depend only on the input length n. Then it is shown that any oblivious T.M. can have its operation on an input of length n simulated by a number of gates proportional to its running time.

This remark has many applications [Paul 1976]. As an example, Fischer and Meyer [1971] have shown that Strassen's fast integer matrix multiplication algorithm can be used to multiply two Boolean $n\times n$ matrices in time $O(n^{\log_2 7}\log n \log\log n \log\log\log n)$ on a Turing Machine. Using the oblivious construction of a circuit simulating the T.M., one thus obtains a B_2-circuit for matrix multiplication with $O(n^{2.82})$ gates. (Recall that Paterson [1975] and Mehlhorn and Galil [1976] have shown that any M-circuit for this problem must have $\Omega(n^3)$ gates.)

While Fischer's result is nice as a technical tool, it is obvious by the earlier remarks that the converse result does not hold. Schnorr [1976d] has strengthened the result by considering oracle T.M.'s in which one work tape initially holds some string A in $\{0,1\}^*$. Define, for $f:\{0,1\}^* \to \{0,1\}$,

$TC(f)(n) = \min\{|M|\cdot T^A(n)\cdot\log S^A(n) \mid M$ with oracle A computes $f\}$ where M ranges over all oracle T.M.'s (see [Schnorr 1976d] for conventions , T^A and S^A are the time and space required by M, and $|M|$ is the number of instructions in program M. Then there exists some fixed polynomial p and a constant c such that

$$C_{B_2}(f) \le c\cdot TC(f) \text{ and } TC(f) \le p(C_{B_2}(f)).$$

Section E. Miscellaneous Notations

Frequently we will be concerned with the rate of growth of a numeric function rather than its exact value. Suppose f and g

are functions defined on the natural numbers. Then we write

$f = O(g)$ iff there is a constant $c > 0$ such that $\limsup_{n\to\infty} (f(n)/g(n)) \le c$

(i.e. if $f(n) \le c\, g(n)$ for all sufficiently large n),

$f = o(g)$ iff $\lim_{n\to\infty} f(n)/g(n) = 0$,

$f = \Omega(g)$ iff there is a constant $c > 0$ such that $\limsup_{n\to\infty} (f(n)/g(n)) \ge c$

(i.e. if $f(n) \ge c\, g(n)$ for all sufficiently large n),

$f = \Theta(g)$ iff $f = O(g)$ and $f = \Omega(g)$,

$f \sim g$ iff $\lim_{n\to\infty} f(n)/g(n) = 1$,

and

$f \lesssim g$ iff $\limsup_{n\to\infty} f(n)/g(n) \le 1$.

If $f = \Theta(g)$ we say that f is *proportional* to g, if $f \sim g$ we say that f is *asymptotically equal* to g, and if $f \lesssim g$ we say that f is *asymptotically less than* g.

If A is any finite set, then $|A|$, *the cardinality* of A, is the number of elements in A. A *partition* of A is a collection of pairwise disjoint sets $B_1, B_2, \ldots, B_k$ such that $\bigcup_{i=1}^{k} B_k = A$. If A and B are sets, then $A-B = \{x \mid x \in A \text{ and } x \notin B\}$.

If X is a set of variables, an *X-variable* is a member of X.

$\mathbb{N}$ is the set of natural numbers, and R is the set of real numbers; $R^* = \{r \in R \mid r \ge 0\} \cup \{\infty\}$.

If $k \le \ell$ are natural numbers, the set of consecutive natural numbers $\{k, k+1, k+2, \ldots, \ell-1, \ell\}$ is denoted $[k:\ell]$.

If x is a real number, then $\lfloor x \rfloor$ is the greatest integer less than or equal to x; $\lceil x \rceil$ is the least integer greater than or equal to x.

"Iff" is an abbreviation for "if and only if", and "w.l.o.g." is an abbreviation for "without loss of generality".

ϕ is the empty set.

If k and ℓ are natural numbers, then $\binom{k}{\ell} = \frac{k!}{\ell!(n-\ell)!}$.

CHAPTER 3

Worst-Case Values for the Complexity of Quadratic Functions

We begin by considering several complexity measures for monotone Boolean functions, and examine the complexity of quadratic m.b.f.'s under these measures. Recall that a monotone function $f \in B_{n,m}$ is quadratic iff for each $i \in [1:m]$, component f_i of f has a set of prime implicants $PI(f_i)$ equal to $\{x_j \cdot x_k \mid \{j,k\} \in A_i\}$, where A_i is some collection of subsets of $[1:n]$ of cardinality two; equivalently, $f_i(x_1,\ldots,x_n) = \bigvee_{\{j,k\} \in A_i} (x_j \cdot x_k)$. The "threshold two" function of n variables T_2^n is quadratic since $PI(T_2^n) = \{x_j \cdot x_k \mid j,k \in [1:n], j \neq k\}$. Boolean matrix multiplication and Boolean convolution are additional examples.

In this chapter we primarily consider the measures MC, $MC_\wedge$, and $MC_\vee$ for circuits, and corresponding measures ML, $ML_\wedge$, and $ML_\vee$ for formulas. For any M-circuit N, we have the relationship $MC_\wedge(N) + MC_\vee(N) = MC(N)$. For an M-formula F which does not contain any constant inputs, we have the corresponding equation $ML_\wedge(F) + ML_\vee(F) = ML(F) - 1$*. Hence for an arbitrary monotone function $f \in B_n$ we have the relationships

*The number of gates in an M-formula is one less than the number of occurrences of variable and constant inputs.

$$MC_\wedge(f) + MC_\vee(f) \le MC(f)$$

and

$$ML_\wedge(f) + ML_\vee(f) \le ML(f) - 1,$$

but at least the first inequality may be strict for specific functions f. For example, in Chapter 5 we will show that $MC_\wedge(T_2^n) = \lceil \log n \rceil$ and $MC_\vee(T_2^n) = 2n-4$, but will also show that if n is a power of 2, any M-circuit for T_2^n which has exactly $\lceil \log n \rceil$ $\wedge$-gates must also have at least $2n + 3\lceil \log n \rceil - 9$ $\vee$-gates, and hence any M-circuit for T_2^n must have at least one more gate than the sum $MC_\wedge(T_2^n) + MC_\vee(T_2^n)$ for these values of n.

Two areas in the complexity of single-output quadratic m.b.f.'s are presented in this chapter. The asymptotic value of the largest complexity for all quadratic m.b.f.'s is explored first. It is shown that every quadratic m.b.f. in B_n has an M-circuit complexity of $O(n^2/\log n)$. That this upper bound is within a constant factor of the best possible result is shown by proving that "most" quadratic m.b.f.'s have circuit complexity over the larger basis B_2 of $\Omega(n^2/\log n)$.

The other area is the determination of the number of $\wedge$-gates necessary and sufficient to compute any quadratic m.b.f. in B_n. We show that n-1 $\wedge$-gates are sufficient in any M-formula, and exhibit a quadratic m.b.f. for which $\lfloor 2/3n \rfloor$ $\wedge$-gates are necessary in any M-circuit. By restricting the types of M-circuits allowed, we can demonstrate a quadratic function for which $n - o(n)$ $\wedge$-gates are necessary in any restricted M-circuit, due to an observation by V.Chvatal.

Section A. The Total Number of Gates

We first demonstrate closely matching upper and lower bounds for the worst-case complexity of quadratic monotone Boolean functions.

Definition: Suppose $n \in \mathbb{N}$. Define

$$QC(n) = \max\{MC(f) | f \text{ is a quadratic m.b.f. in } B_n\},$$

$$QC_\wedge(n) = \max\{MC_\wedge(f) | f \text{ is a quadratic m.b.f. in } B_n\},$$

and $$QC_\vee(n) = \max\{MC_\vee(f) | f \text{ is a quadratic m.b.f. in } B_n\}.$$

We similarly define $QL(n)$, $QL_\wedge(n)$, and $QL_\vee(n)$.

We can use standard counting techniques as found in numerous proofs [Shannon 1949, Lupunov 1962, Fischer 1974] to establish lower bounds on QC.

3.1 Theorem: Suppose $\varepsilon > 0$. Then most quadratic m.b.f.'s f in B_n have

$$C_{B_2}(f) \geq (1 - \varepsilon)n^2/(4 \log n)$$

for n sufficiently large, where B_2 is the complete basis of all two-input Boolean functions.

Proof: Let $Q_n \subseteq B_n$ denote the set of all quadratic m.b.f.'s of n variables. Since there are $\binom{n}{2}$ subsets of the variables $\{x_1, x_2, \dots x_n\}$ with two members, $|Q_n| = 2^{\binom{n}{2}} - 1$.

Suppose $f \in Q_n$. We know that there is some B_2-circuit N which computes f since B_2 is a complete basis. If N contains two gates which compute the same function, then as in Chapter 2, one of these may be eliminated to yield a B_2-circuit with fewer gates which also computes f. Also, if f is not a constant function, it is possible to construct a B_2-circuit for f in which each constant node has outdegree zero. We thus consider only B_2-circuits in which each gate computes a distinct function, and in which constant nodes have outdegree zero; we call such circuits reduced.

Now suppose $q \in \mathbb{N}$ and N is a reduced B_2-circuit of size $\leq q$ which computes a function $f \in Q_n$. By possibly adding additional "dummy" gates, we may assume that there is a reduced B_2-circuit of size exactly q in which one of the gates computes f. Let R_q denote the number of quadratic m.b.f.'s computed by such reduced circuits of size q. Since each gate has two inputs, and can be labelled with any of the 16 binary Boolean functions, there are at most $16(q + n)^2$ possible assignments of function-input pair combinations for each gate in a reduced B_2-circuit, giving a total of at most $[16(q + n)^2]^q$ such assignments of reduced B_2-circuits. However, many of these labellings describe the same circuit -- in fact, there are q! different enumerations of the computational nodes of any fixed reduced circuit. Hence, there are at most $16^q(q + n)^{2q}/q!$ reduced circuits of size q. Since the (quadratic) output function may be computed at any gate of the circuit, we have

$$R_q \le q \cdot 16^q\,(q+n)^{2q}/q!$$

$$\le (16^q(q+n)^{2q+1}\cdot e^q)/(\sqrt{2\pi q}\cdot q^q) \quad \text{(using Stirling's approximation)}$$

$$\le \sqrt{(2/\pi)}\,(64e)^q\cdot q^{q+(1/2)} \quad \text{(when } q \ge n\text{)}$$

$$\le (cq)^q. \quad \text{(for some } c \ge 0 \text{ and } q \text{ sufficiently large)}$$

Now, suppose that $q \le (1-\varepsilon)\binom{n}{2} / \log \binom{n}{2}$. Then

$$R_q \le [\,c\,(1-\varepsilon)\tbinom{n}{2} / \log \tbinom{n}{2}]^{(1-\varepsilon)\binom{n}{2} / \log \binom{n}{2}}$$

$$\le \tbinom{n}{2}^{(1-\varepsilon)\binom{n}{2} / \log \binom{n}{2}} \quad \text{(when } \log \tbinom{n}{2} \ge c(1-\varepsilon)\text{)}$$

$$\le 2^{(1-\varepsilon)\binom{n}{2}} .$$

Hence the fraction of all quadratic m.b.f.'s counted in R_q is

$$R_q/2^{\binom{n}{2}} \le 2^{(1-\varepsilon)\binom{n}{2}}/2^{\binom{n}{2}} = 2^{-\varepsilon\binom{n}{2}}$$

which tends to 0 as $n \to \infty$. Thus most quadratic functions have B_2 complexity at least $(1-\varepsilon)\binom{n}{2} / \log \binom{n}{2} \sim (1-\varepsilon)(n^2)/(4 \log n)$.

□ Theorem 3.1

3.2 Corollary: $QC(n) = \Omega(n^2/\log n)$.

Proof: For any quadratic m.b.f. f, $MC(f) \ge C_{B_2}(f)$ since

$M \subseteq B_2$. □ Corollary 3.2

We can prove that all quadratic m.b.f.'s can be computed using a monotone circuit which is no more than a small constant factor larger than this bound. The idea of the proof is similar to that used in the Four Russian's algorithm for transitive closure [Arlazarov et al. 1970], and has been used previously for constructing efficient circuits for other classes of functions [Tarjan 1976].

3.3 Theorem: Suppose $f \in B_n$ is a monotone quadratic Boolean function. Then

$$MC(f) \le 4\ n^2/\log n \ + O(n).$$

Proof: Let $m = \lfloor \log n \rfloor$. We partition the input variables of f into $\lceil n/m \rceil$ subsets, the i^{th} subset S_i consisting of the set $S_i = \{x_j \mid (i-1)m < j \le i \cdot m,\ j \le n\}$ for $i \in [1:n/m]$. Each subset, with the possible exception of the last, has m variables in it. If $f \in Q_n$ is given, for each input variable x_i and each subset S_j we define

$$A_{ij} = \{x_k \in S_j \mid k \ge i \text{ and } x_i x_k \in PI(f)\}.$$

Note that $A_{ij} = \emptyset$ for $j < \lceil i/m \rceil$. Let $\hat{A}_{ij}$ be the Boolean function of the disjunction of all variables in A_{ij} (recall that by convention $\hat{A}_{ij}$ is the constant function $\underline{0}$ when $A_{ij} = \emptyset$). We clearly can express f as the factorization

$$f(x_1,\ldots,x_n) = \bigvee_{i=1}^{n} (x_i \wedge (\bigvee_{j = \lceil i/m \rceil}^{\lceil n/m \rceil} \hat{A}_{ij})) \qquad (1)$$

Our goal will be to compute the functions $\hat{A}_{ij}$ efficiently, and use the decomposition (1) to compute f. We make use of the following result [Tarjan 1976] which allows efficient computation of sets of disjunctions.

<u>3.4 Lemma</u>: Suppose X is a set of m variables, and $f_1, f_2, \ldots, f_k$ are monotone Boolean functions which are the disjunction of subsets of variables of X. Then there exists a circuit N consisting solely of v-gates which computes $F = \{f_1, f_2, \ldots, f_k\}$ such that $C_{\{v\}}(N) \le 4km/\log k$.

<u>Proof of lemma</u>: Let $r = \lfloor \log k \rfloor$. We decompose the set of X variables into $\lceil m/r \rceil$ subsets $T_1, T_2, \ldots, T_{\lceil m/r \rceil}$ each containing no more than r variables. For each i, the circuit N initially computes the disjunction of all possible subsets of T_i. This can be done in a manner which has one disjunction for each subset of size ≥ 2 by computing the disjunction of smaller subsets first, and then constructing the disjunction of a larger set with one additional variable by using a single v-gate to add this variable to the previously constructed disjunction of the smaller set. Hence, this can be performed using at most $2^r - r - 1$ disjunctions for each $i \in [1:\lceil m/r \rceil]$, giving a total of

$(2^r - r - 1)(\lceil m/r \rceil) < k\lceil m/r \rceil$ disjunctions. Since each function f_j can be constructed by computing the disjunction of $\lceil m/r \rceil$ of these previously computed functions (one per subset T_i), we can compute F with at most an additional $k(\lceil m/r \rceil - 1)$ v-gates. Since $\lceil m/r \rceil \le 2m/\log k$, the total number of v-gates used in this construction is at most $2k\lceil m/r \rceil - k \le 4km/\log k$.

□ Lemma 3.4

Returning to the proof of Theorem 3.3, suppose that j is fixed. Note that $A_{ij} \neq \emptyset$ for at most $j \cdot m$ values of i, namely for each $i \in [1:j \cdot m]$. Hence the computation of the functions A_{ij} for all $j \cdot m$ values of i can be accomplished using the method of Lemma 3.4 with at most $4jm^2/\log(jm)$ v-gates. Thus, one can compute all functions A_{ij} for $i \in [1:n]$, $j \in [1:\lceil n/m \rceil]$ using at most

$$\sum_{j=1}^{\lceil n/m \rceil} (4jm^2)/(\log(jm)) = 4m^2\left(\sum_{j=1}^{\lceil n/m \rceil} j/\log(jm)\right)^{*}$$

$$= 4m^2[n^2/(2m^2 \log n) + O(n/\log^2 n)]$$

$$= 2n^2/\log n + O(n) \qquad \text{v-gates.}$$

[*] To sum the series, note that

$$\sum_{j=1}^{\lceil n/m \rceil} j/\log(jm) \le \int_2^{n/m} (x/\log(xm))\,dx + 2n/(m \log n)$$

$$\le (1/m^2) \int_2^{n} x/\log x \; dx + 2n/(m \log n)$$

(cont. on next page)

Finally, combining these disjunctions to compute f using (1) can be done using an additional $n - 1$ $\wedge$-gates (since $\hat{A}_{nj} = \underline{0}$ for all j); the number of additional $\vee$-gates is at most

$$\begin{aligned}\sum_{i=1}^{n}(\lceil n/m\rceil - \lceil i/m\rceil) + n - 1 &\leq \sum_{i=1}^{n}((n-i)/m + 1) + n - 1 \\ &= (\sum_{j=0}^{n-1} j)/m + 2n - 1 \\ &= n^2/2m + O(n).\end{aligned}$$

Again, since $n^2/m \leq 2n^2/\log n$, we have a total of $4n^2/\log n + O(n)$ gates in the M-circuit constructed.

□ Theorem 3.3

Section B. Asymptotic Bounds on $\wedge$-gates

In the remainder of this chapter, we consider the measures $QC_\wedge$ and $QL_\wedge$ which are the minimal number of $\wedge$-gates required in any

But

$$\begin{aligned}\int_2^n x/\log x\, dx \leq \sum_{i=2}^{n} i/\log i &\leq \sum_{i=2}^{\lfloor n/2\rfloor + 1} [i/\log i + (n-i+2)/\log(n-i+2)] \\ &\leq \sum_{i=2}^{\lfloor n/2\rfloor+1} 2(\lceil n/2\rceil+1)/\log(n/2 + 1) \\ &= 2(\lfloor n/2\rfloor(\lceil n/2\rceil+1))/(2 \log (\lceil n/2\rceil+1)) \\ &= n^2/2 \log n + O(n/\log n).\end{aligned}$$

circuit or formula respectively to express a quadratic m.b.f. of n variables. We do not consider the measures $QC_\vee$ and $QL_\vee$, and consider determination of their values to be an interesting open question. When arbitrary M-circuits are allowed, the bounds we obtain on $\wedge$-complexity are not as tight as those in the previous section, but tighter bounds are given in Section C when $\wedge$-gates are not allowed to have a path to another $\wedge$-gate in the circuit.

3.5 Lemma: Suppose $n \geq 2$. Then $QC_\wedge(n) \leq QL_\wedge(n) \leq n-1$.

Proof: Obviously $QC_\wedge(n) \leq QL_\wedge(n)$ since a formula may be considered a special kind of a circuit.

We prove that $QL_\wedge(n) \leq n-1$ by induction on n.

If $n = 2$, then clearly $QL_\wedge(2) = 1$ since the only quadratic m.b.f. of 2 variables is $T_2^2(x_1,x_2) = x_1 \wedge x_2$.

Now assume inductively that $QL_\wedge(n) \leq n-1$, and suppose that f is a quadratic m.b.f. of $n+1$ variables. We can express f as the factorization

$$f(x_1,\dots,x_{n+1}) = (x_{n+1} \wedge g(x_1,\dots,x_n)) \vee h(x_1,\dots,x_n)$$

where $g = \bigvee\{x_i \mid x_i \cdot x_{n+1} \in PI(f)\}$ and

$h = \bigvee\{x_i \cdot x_j \mid x_i \cdot x_j \in PI(f),\ i \neq n+1,\ j \neq n+1\}$

Clearly $ML_{\wedge}(g) = 0$ since it is simply a disjunction of single variables. Also $ML_{\wedge}(h) \leq QL(n)$ since h is a quadratic m.b.f. of n variables. Inductively, we then know that $ML_{\wedge}(h) \leq n-1$, so $ML_{\wedge}(f) \leq ML_{\wedge}(g) + ML_{\wedge}(h) + 1 \leq n$.

□ **Lemma 3.5**

Remark: One can show that $QL_{\wedge}(4) = 2$ by examining all quadratic m.b.f.'s of 4 variables, and hence a similar proof demonstrates that $QL_{\wedge}(n) \leq n-2$ for $n \geq 4$.

We would now like to determine whether there are quadratic m.b.f.'s which require a number of $\wedge$-gates which is close to the upper bound provided in Lemma 3.5.[†] It is easy to demonstrate a quadratic m.b.f. in B_n for which $\Omega(n)$ $\wedge$-gates are required. For example, suppose n is even and consider the function $f_0(x_1,\ldots,x_n) = (x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee \ldots \vee (x_{n-1} \wedge x_n)$; the proof that

[†] We remark that the number of prime implicants of a quadratic m.b.f. correlates poorly with its $\wedge$-complexity since (as we show in Chapter 5) the function T_2^n, which has $\binom{n}{2}$ prime implicants, only requires $\lceil \log n \rceil$ $\wedge$-gates to compute.

f_0 requires $n/2$ $\wedge$-gates to compute is relatively straightforward. It is an open question whether $QC_\wedge(n)$ is asymptotic with n. Our objective is to show that the constant factor of the trivial lower bound of $n/2$ may be improved to $2/3$.

<u>3.6. Theorem</u>: Suppose $n \geq 2$. There is a quadratic m.b.f. $f \epsilon B_n$ such that $MC_\wedge(f) = \lfloor 2n/3 \rfloor$.

<u>Proof</u>: We define f as follows: partition the n variables into $k = \lceil n/3 \rceil$ subsets $S_1, S_2, \ldots, S_k$, each containing 3 variables (except possibly S_k). We consider the function

$$f(x_1,\ldots,x_n) = T_2^3(S_1) \vee T_2^3(S_2) \vee \ldots \vee T_2^3(S_k) \qquad (2)$$

where $T_2^3\,(S_i)$ is the threshold 2 function of the variables in S_i (for the set S_k, if n is not evenly divisible by 3, we consider there to be 1 or 2 "dummy" variables set to the constant 0). We claim that $MC_\wedge(f) = \lfloor 2n/3 \rfloor$, and prove this result by induction on k in the event $n \equiv 0 (\text{mod } 3)$. The other cases $n \equiv 1(\text{mod } 3)$ and $n \equiv 2(\text{mod } 3)$ are proven similarly. We first show that $MC_\wedge(f) \geq 2n/3$.

Suppose $k = 1$ and $n = 3$. Then $f(x_1,x_2,x_3) = T_2^3(x_1,x_2,x_3)$. By Theorem 5.9 in which it is shown using other means that $MC_\wedge(T_2^n) = \lceil \log n \rceil$, we have $MC_\wedge(f) = \lfloor \log_2 3 \rfloor = 2$.

Now assume the statement true for k and $n = 3k$ -- namely that the function

$$f(x_1,\ldots,x_n) = T_2^3(S_1) \vee \ldots \vee T_2^3(S_k)$$

has $MC_\wedge(f) \geq 2k$. We are concerned with the function

$$f'(x_1,\ldots,x_n,y_1,y_2,y_3) = f(x_1,\ldots,x_n) \vee T_2^3(y_1,y_2,y_3)$$

where we have re-named the variables for ease of description.

Suppose N is an M-circuit computing f'. The general aim of the proof will be to show that there are constants a_1,a_2 and a_3 in $\{0,1\}$ with the following property: if y_i is set to a_i (for $i = 1,2,$ and 3), then the resulting circuit N' computes f; moreover, it is possible to eliminate two unnecessary $\wedge$-gates from N' since each gate has at least one predecessor which computes the constant function $\underline{1}$. We proceed in a fashion similar to Paterson [1976].

<u>Definition</u>: Suppose N is a circuit which computes f', and suppose $i,j \in \{1,2,3\}$. We say a node G of N has property $P_{i,j}$ iff $y_i\, y_j$ is a prime implicant of $\mathrm{Res}(G,N)$.

<u>Definition</u>: If P is any property on nodes of N, we define the <u>initial occurrences</u> of P, denoted $I(P)$, to be

$$I(P) = \{G \in \mathrm{Nodes}(N) \mid G \text{ satisfies } P\text{, but no predecessor of } G \text{ satisfies } P\}.$$

<u>3.7 Lemma</u>: Suppose N is an M-circuit which computes f', and $i,j\epsilon\{1,2,3\}$, $i \neq j$. Then

(1) $I(P_{i,j}) \neq \emptyset$

and (2) $I(P_{i,j})$ consists solely of $\wedge$-gates.

<u>Proof</u>: Since the input variables do not satisfy $P_{i,j}$ and the output gate of N does, by passing up the circuit N from the output gate we must eventually find a gate which satisfies $P_{i,j}$ but none of whose predecessors do. This establishes statement (1). For statement (2), suppose that G is an $\vee$-gate which satisfies $P_{i,j}$, and let $\text{Pred}(G,N) = \{J,K\}$. By the remarks in Chapter 2 on prime implicants, we know that $PI(G,N) \subseteq PI(J,N) \cup PI(K,N)$, and since $y_i y_j \epsilon PI(G,N)$, one of J or K must satisfy $P_{i,j}$. Hence $G \not\in I(P_{i,j})$.

□ Lemma 3.7

<u>3.8 Lemma</u>: Suppose N,i, and j are as above, and $G\epsilon I(P_{i,j})$. If $\text{Pred}(G) = \{J,K\}$, then either $y_i \epsilon PI(J,N)$ and $y_j \epsilon PI(K,N)$, or vice versa.

<u>Proof</u>: By Lemma 3.7, if $G\epsilon I(P_{i,j})$, then G is an $\wedge$-gate. Every prime implicant of G is a product of a member of $PI(J)$ and

a member of PI(K). Since neither PI(J) nor PI(K) contains $y_i \cdot y_j$, we must have either $y_i \epsilon PI(J)$ and $Y_j \epsilon PI(K)$, or vice versa.

□ Lemma 3.8.

<u>3.9 Lemma</u>: If N is as above, then $I(P_{1,2}) \cup I(P_{2,3}) \cup I(P_{1,3}) \geq 2$.

<u>Proof</u>: Suppose to the contrary that there is only a single gate G in $I(P_{1,2}) \cup I(P_{2,3}) \cup I(P_{1,3})$, and suppose $Pred(G,N) = \{J,K\}$. By Lemma 3.8, we may suppose w.l.o.g. that $y_1 \epsilon PI(J,N)$ and $y_2 \epsilon PI(K,N)$ since $G \epsilon I(P_{1,2})$. Again by virtue of Lemma 3.8 and the fact that $G \epsilon I(P_{2,3})$, we must have $y_3 \epsilon PI(J,N)$ since $y_2 \epsilon PI(K,N)$ implies that $y_2 \epsilon PI(G,N)$, and hence that $y_1 y_2 \not\epsilon PI(G,N)$, contrary to the assumption that $G \epsilon I(P_{1,2})$. But then $G \not\epsilon I(P_{1,3})$ since $y_1 y_3$ cannot be a prime implicant of $Res(G,N) = (y_1 \vee y_3 \vee h_1) \wedge (y_1 \vee h_2)$. This contradiction implies that the union contains at least 2 gates.

□ Lemma 3.9.

We are now in a position to prove the main theorem. Since there are at least two distinct $\wedge$-gates in $I(P_{1,2}) \cup I(P_{2,3}) \cup I(P_{1,3})$ and since none of these sets is empty by Lemma 3.7, we may w.l.o.g. assume that there exist distinct gates G and H in N such that $G \epsilon I(P_{1,2})$ and $H \epsilon I(P_{2,3})$. Assume using Lemma 3.8 that

$\mathrm{Pred}(G,N) = \{K_1,K_2\}$ and $y_2 \in \mathrm{PI}(K_1,N)$; and similarly that $\mathrm{Pred}(H,N) = \{K_3,K_4\}$ and $y_2 \in \mathrm{PI}(K_3,N)$. (It may be possible that some of the K nodes are not distinct.) The assignment of the variables $y_1 = 0$, $y_2 = 1$, and $y_3 = 0$ results in a circuit N' which computes the function f. Moreover, in N' the inputs K_1 and K_3 to G and H respectively now compute the constant function $\underline{1}$, and hence gates G and H may be eliminated in N' since $\underline{1} \wedge g = g$ for any Boolean function g. Hence if the resulting circuit is called N'', we have

$$MC_\wedge(N) \geq MC_\wedge(N'') + 2 \geq MC_\wedge(f) + 2 = 2n + 2$$

by induction, and the lower bound is complete.

To show that $C_\wedge(f) \leq \lfloor(2/3)n\rfloor$, it suffices to observe that

$$T_2^3(x_1,x_2,x_3) = ((x_1 \vee x_2) \wedge x_3) \vee (x_1 \wedge x_2)$$

and hence $C_\wedge(T_2^3) \leq 2$. Thus using the definition (2) one can compute f using $\lfloor(2/3)n\rfloor$ $\wedge$-gates.

□ Theorem 3.6

__3.10 Corollary__: Suppose $n \geq 4$. Then

$$\lfloor(2/3)n\rfloor \leq QC_\wedge(n) \leq QL_\wedge(n) \leq n-2 .$$

Section C. A Graph Problem

We close this chapter by describing a graph-theoretic problem which is related to the $\wedge$-complexity of quadratic m.b.f.'s when they are computed with a restricted class of M-circuits. The research in the remainder of this chapter was done in conjunction with Ronald Rivest. We introduce notation similar to Harary [1969] for undirected graphs.

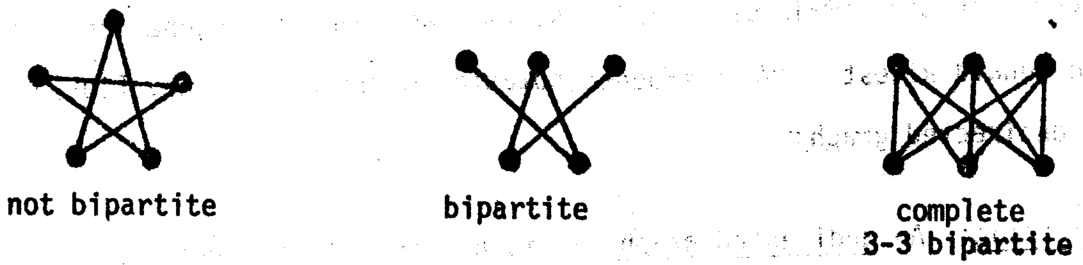
Definition: An *undirected graph* G is a pair (V,E), where V is a finite set whose members are called *nodes* or *vertices*, and $E \subseteq \{B \subseteq V \mid |B| = 2\}$ is a set whose members are called *edges*.

A *bipartite graph* is a graph $G = (V,E)$ whose vertices can be partitioned into two non-empty sets V_1 and V_2 such that if edge $\{v_1,v_2\}\in E$, then $v_1\in V_1 \Leftrightarrow v_2\in V_2$; that is, each edge consists of one point in V_1 and one point in V_2.

A *complete* *k-ℓ* *bipartite graph* is a bipartite graph $(V = V_1\cup V_2,E)$ such that $|V_1| = k$ $|V_2| = \ell$, and $E = \{\{v_1,v_2\} \mid v_1\in V_1,\ v_2\in V_2\}$.

Definition: If $\mathcal{G} = \{(V_1,E_1),(V_2,E_2),\ldots,(V_k,E_k)\}$ is a collection of undirected graphs, and $G = (V,E)$ is an undirected graph, then an *exact covering* C of G by $\mathcal{G}$ is a collection of one-to-one mappings $h_i:V_i \to V$ for $i\in[1:k]$ such that $E = \bigcup_{i\in[1:k]}\{h(e) \mid e\in E_i\}$

where $h(\{a,b\}) = \{h(a),h(b)\}$ for any edge $\{a,b\}\epsilon E_i$. The cost of the cover, denoted $\hat{D}(C)$, is k.



not bipartite

bipartite

complete 3-3 bipartite

Fig. 3.1 Some Graphs
Points represent nodes and lines represent edges.

We can now define a measure on graphs.

Definition: Suppose $n\epsilon\mathbb{N}$, and G is an n-node undirected graph. Then

$$D(G) = \min\{\hat{D}(C) \mid C \text{ is an exact cover of } G \text{ by } \{G_1,\ldots,G_k\}, \text{ where } G_i \text{ is a complete bipartite graph for } i\epsilon[1:k]\} .$$

Define $H(n) = \max\{D(G) \mid G \text{ is an } n\text{-node undirected graph}\}$.

The connection between quadratic m.b.f.'s and graphs is as follows: Suppose f is an n-variable Boolean function. We can associate a corresponding n-node undirected graph $G_f = (V,E)$, where $V = \{1,2,\ldots,n\}$ and $E = \{\{i,j\} \mid x_i x_j \epsilon PI(f)\}$. Conversely, suppose $G = (V,E)$ is an n-node undirected graph. By possibly renaming the vertices, we may assume $V = [1:n]$. Define the Boolean function $f_G:\{0,1\}^n \to \{0,1\}$ by

$$f_G(x_1,\ldots,x_n) = \vee\{x_i \cdot x_j \mid \{i,j\}\epsilon E\} .$$

The functions $f \to G_f$ and $G \to f_G$ clearly demonstrate a one-to-one correspondance between the set of quadratic m.b.f.'s in B_n and the set of n-node undirected graphs with at least one edge. See Figure 3.2 for some examples.

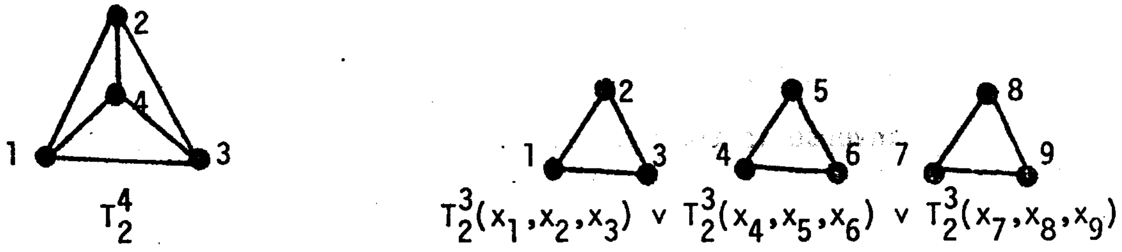

2
4
1
3
T^4_2
2
1
3
5
4
6
7
8
9
$T^3_2(x_1,x_2,x_3) \vee T^3_2(x_4,x_5,x_6) \vee T^3_2(x_7,x_8,x_9)$

Fig. 3.2 Some Functions and their Graphs

In fact, we can get a more strict correspondence in mathematical terms.

Notation: $\hat{Q}_n = \{f \mid f$ is a quadratic m.b.f. of n variables$\} \cup$ $\{\underline{0}$, the constant function 0 of n variables$\}$.

$\mathcal{G}_n = \{G = (V,E) \mid G$ is an undirected graph, and $V = [1:n]\}$.

Z_n is the graph $([1:n],\emptyset)$ in $\mathcal{G}_n$.

If $G_1 = (V,E_1)$ and $G_2 = (V,E_2)\in\mathcal{G}_n$, define $G_1\cup G_2$ to be the graph $(V,E_1\cup E_2)$.

If A is any set, and $f{:}A \to B$, then $f(A) = \{f(a) \mid a\in A\}$.

If $G = (V,E)$ is an undirected graph, and $h{:}V \to W$, then $h(G)$ is the graph $(h(V),h(E))$.

3.11 Lemma: Suppose $n \geq 0$. Then

(1) $(\hat{Q}_n,\vee,\underline{0})$ and $(\mathcal{G}_n,\cup,Z_n)$ are monoids,

and (2) the function $f \to G_f$ is a monoid isomorphism whose inverse is the map $G \to f_G$.

Proof: Statement (1) is easily checked, as are the facts that the maps in (2) are one-to-one, onto, and inverses of each other. Moreover, if f_1 and $f_2\in\hat{Q}_n$, then

$$PI(f_1\vee f_2) = PI(f_1) \cup PI(f_2)$$

since each prime implicant is the product of two variables. Thus, if

$G_{f_1 \vee f_2} = ([1:n],E)$, $G_{f_1} = ([1:n],E_1)$, and $G_{f_2} = ([1:n],E_2)$, it is easy to check that $E = E_1 \cup E_2$, and the claim is verified.

□ Lemma 3.11

3.12 Corollary: Suppose $C = (h_1,\ldots,h_k)$ is a covering of the graph G by graphs $G_1,G_2,\ldots,G_k$. Then

$$f_G = f_{h_1(G_1)} \vee f_{h_2(G_2)} \vee \cdots \vee f_{h_k(G_k)}.$$

Proof: Proveń inductively using Lemma 3.11.

□ Corollary 3.12

We now connect the complexity of functions and their graphs. We will say an M-circuit or M-formula N is single-level if there is no path between any pair of $\wedge$-gates in N. Obviously, any quadratic m.b.f. can be computed by a single-level M-formula and hence a single-level M-circuit. In a single-level M-circuit N, each immediate predecessor of every $\wedge$-gate consists of the disjunction of some set of input and constant nodes. Moreover, if N computes a quadratic m.b.f. f, then there is some subset α of the $\wedge$-gates of N such that $f = \underset{A\in\mathcal{a}}{\vee} Res(A,N)$. Thus, given a single-level M-circuit N computing a quadratic m.b.f. f, one may construct a single-level M-formula F which also computes f and which has the same number of $\wedge$-gates as N since computing any number of disjunctions of variables takes no $\wedge$-gates.

If f is a quadratic m.b.f., define

$$SLC_\wedge(f) = \min\{MC_\wedge(N) \mid N \text{ is a single-level M-circuit which computes } f\}.$$

We shall prove

<u>3.13 Theorem</u>: Suppose f is a quadratic m.b.f.. Then $SLC_\wedge(f) = D(G_f)$.

<u>Proof</u>: Suppose f is a quadratic m.b.f., and that $D(G_f) = k$. Let $C = (h_1,\ldots,h_k)$ be an exact cover of G_f by complete bipartite graphs $G_1,\ldots,G_k$. We will construct a single-level M-formula for f which has one $\wedge$-gate for each graph in the cover of G_f. Suppose $G_i = (V_i = V_{i1} \cup V_{i2}, E_i)$ is the decomposition of the vertices of G_i as in the definition of bipartite graphs.

Note that $h_i(G_i)$ is also a complete bipartite graph. Since each graph is complete,

$$f_{h_i(G_i)}(\vec{x}) = \vee\{x_j x_k \mid \{j,k\} \in h_i(E_i)\}$$
$$= (\vee\{x_j \mid j \in h_i(V_{i1})\}) \wedge (\vee\{x_k \mid k \in h_i(V_{i2})\}).$$

Hence, using Corollary 3.12, we can express f as

$$f = f_{G_f} = f_{h_1(G_1)} \vee f_{h_2(G_2)} \vee \cdots \vee f_{h_k(G_k)}$$
$$= \bigvee_{i=1}^{k} [(\vee\{x_j \mid j \in h_i(V_{i1})\}) \wedge (\vee\{x_k \mid k \in h_i(V_{i2})\})]$$

which is a formula with k $\wedge$-gates. Thus $SLC_\wedge(f) \le k$.

Conversely, suppose f is a quadratic m.b.f., and $SLC(f) = k$. By the remarks preceding the theorem, there is a single-level M-formula F for f such that $MC_\wedge(F) = k$. Since F is single-level, we may write

$$F = \bigvee_{i=1}^{k} [(\bigvee_{j \in A_i} x_j) \wedge (\bigvee_{k \in B_i} x_k)]$$

for some subsets A_i and B_i of $[1:n]$ for each $i \in [1:k]$. Since $PI(f)$ has no single variable terms, we know that $A_i \cap B_i = \emptyset$ for each $i \in [1:k]$. Thus

$$f(x) = \bigvee_{i=1}^{k} \{x_j x_k \mid j \in A_i \text{ and } k \in B_i\}.$$

So if we define G_i to be the complete bipartite graph with vertex set $A_i \cup B_i$ (decomposed as A_i and B_i) for each $i \in [1:k]$, we observe that $C = (id, id, \ldots, id)$ is an exact cover of G_f by complete bipartite graphs $G_1, G_2, \ldots, G_k$, where id is the identity map. Hence $D(G_f) \leq k$.

□ Theorem 3.13

If we define $SLQC_\wedge(n)$ to be the maximum value of $SLC(f)$ for qny quadratic m.b.f. of n variables, then Theorem 3.13 implies that $SLQC_\wedge(n) = H(n)$. Since any single-level M-formula is in fact an M-formula, we know $QL_\wedge(n) \leq SLQC_\wedge(n)$. We note that the proof given for Lemma 3.5 actually yields a single-level M-formula for any quadratic m.b.f., so $SLQC_\wedge(n) \leq n-1$. Since $QL\ (n) = \theta(n)$, we know also that $SLQC\ (n) = \theta(n)$. For the latter function, it was pointed out to the author by V. Chvatal[†] that one can actually get asymptotically matching upper and lower bounds on $H(n)$.

<u>3.14 Theorem</u>: $SLQC_\wedge(n) = H(n)$, and $H(n) \sim n$.

We include a sketch of the proof of Theorem 3.14 in Appendix 1.

[†] Private communication, 1977. Recent work by Bermond [1978] has tightened the bound given on $H(n)$.

Section D. Open Questions

At the end of each chapter, we list some open questions pertaining to that chapter.

1. Demonstrate a m.b.f. for which $ML_{\wedge}(f) + ML_{\wedge}(f) < ML(f) - 1$.

2. Is $QL(n) = \Theta(n^2/\log n)$? More generally, do $o(n^2)$ monotone formulas exists for all quadratic m.b.f.'s? $QL(n)$ is obviously $O(n^2)$.

3. What is the asymptotic behavior of $QL_{\vee}(n)$ and $QC_{\vee}(n)$? Determine $QL_{\wedge}$ and $QC_{\wedge}$ more exactly than given in Corcllary 3.10.

4. If f is a quadratic m.b.f., is $SLC_{\wedge}(f) = ML_{\wedge}(f)$? Is $ML_{\wedge}(f) = MC_{\wedge}(f)$?

5. More generally, is $SLQC_{\wedge}(n) = QL_{\wedge}(n)$?

CHAPTER 4

Combinations of Functions

One fundamental problem of Boolean function complexity is determining the relationship between the complexity of individual functions and the complexity of combinations of those functions. The relationship depends not only on the type of combination but also on the model and measure of complexity used. In this chapter, we explore several questions in this area.

Given Boolean functions f and g, we consider primarily the combinations $f \times g$ and $\vee(f \times g)$. Results about the function $\wedge(f \times g)$ may be obtained by duality from those for $\vee(f \times g)$. We do not consider the composition of functions; this area has many interesting open questions. The measures which we consider are formula and circuit size over different bases.

As Paul [1976 p.383] points out, one might expect that $C_{B_2}(f \times g) = C_{B_2}(f) + C_{B_2}(g)$ since the evaluation of f and g on disjoint sets of variables "have nothing to do with each other." This, however, is incorrect; Paul shows that for any $\varepsilon > 0$, there are arbitrarily complex functions f in $B_{n,n}$ such that $C_{B_2}(f \times f) \leq (1 + \varepsilon)C_{B_2}(f)$. In addition, he exhibits arbitrarily complex g in B_n such that $C_{B_2}(\vee(g \times g)) \leq (1 + \varepsilon)C_{B_2}(g)$.

For M-circuits computing monotone functions, the results are drastically different. M. Fischer proved that

$MC(f \times g) = MC(f) + MC(g)$ for any m.b.f.'s $f \in B_{n,k}$ and $g \in B_{m,\ell}$ † [Paul 1976]. It is still unknown whether $MC(\vee(f \times g)) = MC(f) + MC(g) + 1$. In Appendix 2, we present an extension to Fischer's result, due to Galbiati and Fischer, [1978] in which the sets of variables on which f and g depend have one variable in common. In addition, we show that Fischer's result holds when $\vee$-gates and $\wedge$-gates are counted separately; namely that $MC_{\wedge}(f \times g) = MC_{\wedge}(f) + MC_{\wedge}(g)$ and dually $MC_{\vee}(f \times g) = MC_{\vee}(f) + MC_{\vee}(g)$.

We consider the question of formula size additivity; the only combination considered is $\vee(f \times g)$. Here, in contrast to Paul's results on circuit size, we show that formula size is additive under $\vee$, namely

$$L_{\Omega}(\vee(f \times g)) = L_{\Omega}(f) + L_{\Omega}(g) \qquad (1)$$

for any basis Ω which contains $\vee$, and any non-constant Boolean functions $f \in B_n$ and $g \in B_m$. When counting individual types of gates in formulas over the monotone basis M, we show that $ML_{\vee}(\vee(f \times g) = ML_{\vee}(f) + ML_{\vee}(g) + 1$ for arbitrary monotone functions f in B_n and g in B_m; the corresponding question $ML_{\wedge}(\vee(f \times g) \overset{?}{=} ML_{\wedge}(f) + ML_{\wedge}(g)$ remains open, even for simple classes of

†In fact, Fischer shows that any minimal M-circuit for $f \times g$ is composed of disjoint minimal circuits for f and g.

functions such as quadratic functions.

Finally, using fact (1), we demonstrate a gap between the circuit and formula complexity of some specific functions. In particular, we show that there is a sequence of functions whose B_2-circuit complexity is $O(n \log^2 n)$, but whose B_2-formula complexity is $\Omega(n^2/\log \log n)$. A slightly larger gap between these measures for a particular function had been previously shown using different techniques by Paul [1977][†], who observed the implications of fact (1).

The following chart summarizes the current state of what is known about these problems[††].

[†] Worst-case results for these measures show that most functions $f \epsilon B_n$ have $L_{B_2}(f) = \Omega(2^n/\log n)$ and $C_{B_2}(f) = O(2^n/n)$. Hence for most functions the ratio between these two measures is $\Omega(n/\log n)$, but the measures themselves are exponential in n.

[††] Paul's result on the sub-additivity of x in the measure C_{B_2} used multi-output functions.

Combination	Class of Functions	Measure							
		C_{B_2}	MC	$C_\vee$	$C_\wedge$	L_{B_2}	ML	$L_\vee$	$L_\wedge$
$\vee(f\times g)$	B_n	SUB	?	?	?	ADD	ADD	ADD	?
$\times$	B_n	?	ADD	ADD	ADD	-	-	-	-
$\times$	$B_{n,m}$	SUB	ADD	ADD	ADD	-	-	-	-

Table 4.1. Summary of results.

Key - ADD: combination of minimal circuits/formulas gives optimal circuit/formula for combination

SUB: negation of ADD for some particular functions

?: open

-: does not apply

Section A. The Formula Size of $\vee(f\times g)$

In this section we show that one cannot economize when constructing a formula for $\vee(f\times g)$ in an arbitrary binary basis.

4.1 Theorem: Suppose n and $m\in\mathbb{N}$, $f\in B_n$ and $g\in B_m$ are non-constant Boolean functions, and Ω is an arbitrary Boolean basis.

Then

$$L_\Omega(\vee(f\times g)) \geq L_\Omega(f) + L_\Omega(g).$$

Proof: Suppose $F(x_1,\ldots,x_n,y_1,\ldots,y_m)$ is an Ω-formula for $\vee(f\times g)(x_1,\ldots,x_n,y_1,\ldots,y_m) \equiv f(x_1,\ldots,x_n)\vee g(y_1,\ldots,y_m)$. Since f and g are non-constant functions, there are constants $\vec{a}\in\{0,1\}^n$ and $\vec{b}\in\{0,1\}^m$ such that $f(\vec{a}) = g(\vec{b}) = 0$. Thus $F(x_1,\ldots,x_n,\vec{b})$, even when simplified by absorbing constants into gates, is an Ω-formula for $f(\vec{x})\vee g(\vec{b}) = f(\vec{x})$, and hence has at least $L_\Omega(f)$ occurrences of variables from $\{x_1,\ldots,x_n\}$. Similarly, since $F(\vec{a},y_1,\ldots,y_m)$ is a formula for $g(\vec{y})$, F has at least $L_\Omega(g)$ occurrences of variables from $\{y_1,\ldots,y_m\}$. Since the sets $\{x_1,\ldots,x_n\}$ and $\{y_1,\ldots,y_m\}$ are disjoint, F has at least $L_\Omega(f) + L_\Omega(g)$ total occurrences of variables. Since this is true for any formula F for $\vee(f\times g)$, the theorem is proved.

□ Theorem 4.1.

4.2 Corollary: If Ω is any basis containing $\vee$, and f and g are as in Theorem 4.1, then $L_\Omega(\vee(f\times g)) = L_\Omega(f) + L_\Omega(g)$. Particular examples are $\Omega = B_2$ and $\Omega = M$.

To obtain the next corollary, we state without proof the following technical lemma due to Paul [1976].

Notation: For $g \in B_n$, and $k \in \mathbb{N}$, $k > 0$, define $\vee g^k$ by the inductive rules:

$$\vee g^1 = g$$

$$\vee g^k = \vee(g \times (\vee g^{k-1})).$$

(So $\vee g^k$ is the disjunction of k copies of g on disjoint sets of variables.)

4.3 Fact (Paul): There is a constant α such that, for all k and $n \in \mathbb{N}$, $k \geq 1$, and all $g \in B_n$, the following holds:

$$C_{B_2}(\vee g^k) \leq \alpha \cdot \max(n \cdot 2^n, nk \log^2 k) \quad .$$

We also need the following fact due to Krichevskii [1961].

4.4 Fact: There is an $n_0 \in \mathbb{N}$ such that for every $n > n_0$, there is a function $f \in B_n$ such that $L_{B_2}(f) \geq 2^n/(2 \log n)$.

We now can establish the following gap between circuit and formula complexity.

4.5 Corollary (Paul, Bloniarz): There are constants n_0 and α such that if $m > n_0$ is a power of 2 and $n = 2^m/m$, then there is a function $f \epsilon B_n$ such that

$$(1) \quad L_{B_2}(f) \geq (2^{2m})/(2m^2 \cdot \log m)$$

while

$$(2) \quad C_{B_2}(f) \leq \alpha \cdot m \cdot 2^m$$

Hence $L_{B_2}(f) = \Omega(n^2/(\log \log n))$ whereas $C_{B_2}(f) = O(n \log^2 n)$.

Proof: For this proof, we assume that the basis of operators for all formulas and circuits is B_2. Let n_0 be as in Fact 4.4. There is thus a function $g \epsilon B_m$ such that $L(f) \geq 2^m/(2 \log m)$. By induction on Theorem 4.1, it is easy to show that $L(\bigvee g^k) = k \cdot L(g)$, so if we define

$$f = \bigvee(g^{2^m/m^2}),$$

we know that $L(f) \geq (2^m/m^2) \cdot (2^m/(2 \log m))$.

On the other hand, by Fact 4.3, there is a constant α such that

$$C_B(f) \leq \alpha \cdot \max(m \cdot 2^m,\ m \cdot 2^m \cdot (m-2 \log m)^2/m^2$$
$$= \alpha \cdot m \cdot 2^m .$$

The final remark follows since f has $n = (2^m/m^2) \cdot m$ variables and since $m \leq 2 \log n$. □ Corollary 4.5

Section B. Monotone Functions

We now restrict ourselves to the class of single-output monotone Boolean functions and formulas over the basis M. For the combination $\vee(f\times g)$, Corollary 4.2 proved that monotone formula complexity ML is additive since $\vee\in M$. As for counting individual gates, clearly $ML_{\wedge}(\vee(f\times g))\leq ML_{\wedge}(f) + ML_{\wedge}(g)$ for any m.b.f.'s f and g. It remains an open question as to whether there are any non-constant m.b.f.'s f and g for which the inequality is strict. The corresponding question for $ML_{\vee}$ is answered by the following theorem.

4.6 Theorem: Suppose n and $m\in\mathbb{N}$, and $f\in B_n$ and $g\in B_m$ are non-constant m.b.f.'s. Then

$$ML_{\vee}(\vee(f\times g)) = ML_{\vee}(f) + ML_{\vee}(g) + 1$$

Proof: The fact that $ML_{\vee}(\vee(f\times g))\leq ML_{\vee}(f) + ML_{\vee}(g) + 1$ is easily verified by combining an $\vee$-minimal M-formula for f with one for g by an $\vee$-gate. To show the inequality in the opposite direction, suppose that $F(\vec{x},\vec{y})$ is a M-formula for $f(\vec{x})\vee g(\vec{y})$ with the minimal number of $\vee$-gates. We will prove that if H is any $\wedge$-gate of F, then H depends[†] only on variables from the set $X = \{x_1,\ldots,x_n\}$ or only on variables from $Y = \{y_1,\ldots,y_m\}$.

[†] We remark that in a minimal formula, the notions of structural dependence and functional dependence coincide.

Supposing this for the moment, we show that the theorem follows from this claim. Let $\mathcal{H}$ = {H|H is a variable or $\wedge$-gate of F such that $\mathrm{Succ}^+(H,F)$ contains no $\wedge$-gates}. By assumption, no node in $\mathcal{H}$ depends on variables from both X and Y. We look at $\mathrm{Succ}^+(\mathcal{H},F)$. If $\mathrm{Succ}^+(\mathcal{H},F)$ were empty, then since F is a formula there would be a unique node H_0 in $\mathcal{H}$. But then $\mathrm{Res}(H_0,F) = \vee(f\times g)$. Since $\vee(f\times g)$ depends on variables from both X and Y, and by assumption H_0 does not, this is a contradiction. Hence $\mathrm{Succ}^+(\mathcal{H},F)$ is a non-empty collection of $\vee$-gates, one of which is the output gate U (see Figure 3.1(a)); moreover

$$\mathrm{Res}(U,F) = \bigvee_{H\in\mathcal{H}} \mathrm{Res}(H,F) \quad . \qquad (2)$$

We will replace $\mathrm{Succ}^+(\mathcal{H},F)$ with a tree of $\vee$-gates which also realizes the function (2) and obtain a (possibly different) formula for Res(U,F). Let $\mathcal{H}_X$ denote the set $\{H\in\mathcal{H}|H$ depends on variables in X}, and let $\mathcal{H}_Y = \{H\in\mathcal{H}|H$ depends on variables in Y}. By assumption, the pairs of sets $\mathcal{H}_X$ and $\mathcal{H}_Y$ is a partition of $\mathcal{H}$. Form the subformulas

$$F_X = \bigvee_{H\in\mathcal{H}_X} F_H$$

and

$$F_Y = \bigvee_{H\in\mathcal{H}_Y} F_Y$$

(where the disjunction may be associated arbitrarily), and finally the formula $F' = F_X \vee F_Y$. Since $\mathrm{Res}(F) = (\bigvee_{H \in H_X} \mathrm{Res}(H,F)) \vee (\bigvee_{H \in H_Y} \mathrm{Res}(H\ ,F))$, F' defines the same function $\vee(f \times g)$ as F. Moreover, since F was $\vee$-minimal, $\mathrm{Succ}^{+}(H,F)$ contains exactly $|H|-1$ $\vee$-gates;

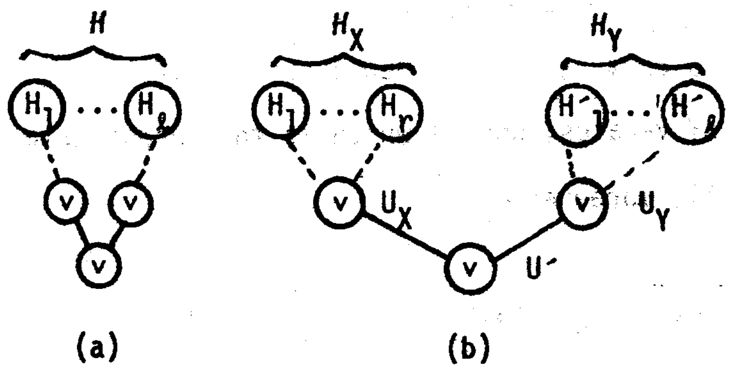


Fig. 4.1 The re-arrangement

since $|H|-1$ $\vee$-gates were added in the reconstruction, F' also contains a minimal number of $\vee$-gates. But $f(x_1,\ldots,x_n) \vee g(y_1,\ldots,y_m) = \mathrm{Res}(F_X)(\vec{x},\vec{y}) \vee \mathrm{Res}(F_Y)(\vec{x},\vec{y})$. Since $\mathrm{Res}(F_X)$ depends only on X and $\mathrm{Res}(F_Y)$ depends only on Y, it is easy to show that $f = \mathrm{Res}(F_X)$ and $g = \mathrm{Res}(F_Y)$. Since the subtrees F_X and F_Y are disjoint, there must be at least $ML_{\vee}(f)$ $\vee$-gates in F_X and $ML_{\vee}(g)$ $\vee$-gates in F_Y, giving a total of $ML_{\vee}(f) + ML_{\vee}(g) + 1$ $\vee$-gates in F', and hence in F.

To show that all $\wedge$-gates in F depend on only one of X or Y, we suppose to the contrary that some $\wedge$-gate depends on both X and Y variables. Let H be a top-most $\wedge$-gate with this property, i.e. H

is an $\wedge$-gate such that H depends on some variable in X and some variable in Y, but any $\wedge$-gate H' in $\mathrm{Pred}^{+}(H,F)$ depends only on X-variables or only on Y-variables. We will use techniques similar to the first portion of the proof to restructure the formula and reduce the number of $\vee$-gates in F.

Suppose that $\mathrm{Pred}(H,F) = \{I,J\}$. Define

$$A(I) = \{G \in \mathrm{Pred}^{*}(I,F) \mid G \text{ is an } \wedge\text{-gate or an input and all intermediate gates (if any) on the path from } G \text{ to } H \text{ are } \vee\text{-gates}\}$$

and define $A(J)$ similarly. Note that neither $A(I)$ nor $A(J)$ is empty by our selection of H. Since the portions of the tree from $A(I)$ to I and from $A(J)$ to J consist solely of $\vee$-gates, there are $|A(I)|-1$ $\vee$-gates in the subtree from $A(I)$ to I and $|A(J)|-1$ $\vee$-gates in the subtree from $A(J)$ to J. Functionally, we have

$$\mathrm{Res}(H,F) = [\bigvee_{G\in A(I)}(\mathrm{Res}(G,F))] \wedge [\bigvee_{G\in A(J)}(\mathrm{Res}(G,F))] \qquad (3).$$

We are now in a position to use Mehlhorn and Galil's lemma (Lemma 2.8) to reduce the number of $\vee$-gates in the formula.

Define

$$A_X = \{G \in A(I) | G \text{ depends on some variable in } X\}$$

$$A_Y = \{G \in A(I) | G \text{ depends on some variable in } Y\}$$

$$B_X = \{G \in A(J) | G \text{ depends on some variable in } X\}$$

and $$B_Y = \{G \in A(J) | G \text{ depends on some variable in } Y\}$$

Observe that since H is a top-most $\wedge$-gate depending on both X and Y, $A_X \cap A_Y = B_X \cap B_Y = \emptyset$. We construct a new formula F' by replacing the subformula F_H in F by a subformula F_0 which realizes the function

$$f_0 = [(\bigvee_{G \in A_X} \mathrm{Res}(G)) \wedge (\bigvee_{G \in B_X} \mathrm{Res}(G))] \vee [(\bigvee_{G \in A_Y} \mathrm{Res}(G)) \wedge (\bigvee_{G \in B_Y} \mathrm{Res}(G))].$$

If none of A_X, A_Y, B_X, nor B_Y is empty, then we define

$$F_0 = [(\bigvee_{G \in A_X} F_G) \wedge (\bigvee_{G \in B_X} F_G)] \vee [(\bigvee_{G \in A_Y} F_G) \wedge (\bigvee_{G \in B_Y} F_G)] \qquad (4)$$

In this case F_0 has at least one fewer $\vee$-gate since the $|A(I)| + |A(J)| - 2$ $\vee$-gates we identified in deriving equation (3) have been replaced by

$$(|A_X|-1) + (|B_X|-1) + (|A_Y|-1) + (|B_Y|-1) + 1 = |A(I)| + |A(J)| - 3$$

$\vee$-gates.

If, for example, A_X alone is empty, then we define F_0 to be the obvious simplification of (4), namely

$$F_0 = (\underset{G\epsilon A_Y}{\vee} G) \wedge (\underset{G\epsilon B_Y}{\vee} G). \qquad (5)$$

In this case, F_0 contains $(|A_Y|-1) + (|B_Y|-1) = |A(I)| + |B_Y|-2$ v-gates. But $B_X \neq \emptyset$ since H depends on both X and Y variables by definition.. Thus $|B_Y| < |A(J)|$ so F_0 has fewer v-gates then F_H.

If one of the other sets is empty we proceed similarly. If two of the sets are empty, then they must differ in their variable sets since H depends on both X and Y variables. So suppose w.l.o.g. that A_X and B_Y are empty. Then $f_0 = \underline{0}$, the constant function zero, so we may let F_0 be the constant 0. In this case the closest v-gate in $\text{Succ}^+(H,F)'$ may be eliminated since one input to the gate is the constant 0; such a gate must exist since the formula F does not compute a constant function.

Hence in all cases F' has at least one fewer v-gate than F. Now if we can show that F' also computes $f(x) \vee g(y)$, then we will have a contradiction. But this is in fact the case. Observe that f_0 implies Res(H,F); in fact, $PI(f_0)$ consists of those monomes of PI(H,F) which consist solely of X-variables or solely of Y-variables. Since $PI(\vee(f\times g))$ has no monomes which contain both variables from X and variables from Y, by Lemma 2.8 we know that each monome in PI(H,F) which contains variables from both X and Y may be eliminated from $PI(H,F)$ without changing the function defined by the formula F. Since the replacement of F_0 for F_H eliminates exactly all such monomes, F and F' define the same function $\vee(f\times g)$. But since F' has at least one fewer $\vee$-gate than F, this is a contradiction to the $\vee$-minimality of F.

□ Theorem 4.6.

Section C. Open Questions

(1) There are several open questions listed in Table 1. Are any solvable in general? If the class of functions is restricted (e.g. to quadratic functions) are any of the questions solvable?

(2) More generally, for arbitrary single-output m.b.f.'s f and g, is

$$MC(f(g(\vec{x}),g(\vec{y}),\ldots,g(\vec{z}))) = MC(f) + MC(g) \quad ?$$

CHAPTER 5

The Monotonic Circuit Complexity of Threshold Functions

One important class of Boolean functions is the set of symmetric functions, those which are invariant under any permutation of their inputs. Formally, a Boolean function $f \epsilon B_{n,m}$ is symmetric iff $f(x_1,x_2,\ldots,x_n) = f(x_{\pi(1)},x_{\pi(2)},\ldots,x_{\pi(n)})$ for every permutation π of the set $[1:n]$. One interesting fact about the symmetric Boolean functions is that they are fairly easy to compute by a circuit over a complete basis. The value of any symmetric function on an argument depends solely on the number of inputs which are set to 1. Since it is possible to construct a linear-sized circuit which counts the number of 1's in an input and outputs that number written in binary, it is possible to use the output of such a "unary to binary" converter to compute the value of the symmetric function on the input. The best known general schemes for computing symmetric functions in this way over the complete basis B_2 require $\sim 6n$ gates in all [Muller and Preparata 1975]†.

As for lower bounds on the complexity of symmetric functions, Lemma 2.11 shows that any non-constant symmetric Boolean function has complexity at least $n-1$ over B_2 since the function depends

† We remark that using a similar idea, one also obtains a fairly small formula (size $O(n^{3.57\ldots})$) for expressing any threshold function [Peterson 1978].

on each of its inputs. Schnorr shows that all but 8 of the symmetric functions of n variables have B_2 circuit complexity of at least $2n-3$ [1974]. Recently, Stockmeyer showed that at least half of the 2^{n+1} symmetric Boolean functions of n variables have B_2 circuit complexity of at least $5n/2 - 5$ [1977]. The latter result, along with an argument of Paul which establishes a similar $5n/2 - O(1)$ lower bound on the complexity of specific functions [1977], are the largest known lower bounds on the B_2 circuit complexity of any single-output Boolean function other than bounds obtained by diagonal arguments [Stockmeyer 1974].[†]

In this chapter, we consider the collection of monotone symmetric Boolean functions, the threshold functions. The main object of study will be the monotone circuit complexity of these functions. Recall the definition of the threshold functions.

<u>Definition</u>: Suppose $k,n \in \mathbb{N}$. Define the Boolean function <u>threshold k of n variables</u>, denoted $T_k^n \in B_n$, by $T_k^n(x_1,\ldots,x_n) = 1$ iff at least k of the inputs $x_1,\ldots,x_n$ are 1.

In particular, T_0^n is the constant function 1 for all values of n, and T_m^n is the constant function 0 for all $m > n$. Since

$$T_1^n(x_i) = \bigvee_{i=1}^{n} x_i \quad \text{and} \quad T_n^n = \bigwedge_{i=1}^{n} x_i,$$

[†] Recently, Schnorr [1978b] has announced a $3n$ - lower bound on the B_2-circuit complexity of a function.

these extremal threshold functions have circuit complexity over M of $n-1$. By observing that

$$T_k^n(x_1,\ldots,x_n) = \neg(T_{n-k+1}^n(\neg x_1, \neg x_2, \ldots, \neg x_n))$$

we know that T_k^n and T_{n-k+1}^n are duals of each other, so that in general $C_{B_2}(T_k^n) = C_{B_2}(T_{n-k+1}^n)$ and $MC(T_k^n) = MC(T_{n-k+1}^n)$ for $k \in [0:n+1]$.

For circuits over B_2, the best known results are the following. The general upper bound of $6n$ for the B_2-circuit complexity of any symmetric function obviously applies to the threshold functions. (This general bound can be improved upon for small values of k.) Stockmeyer's techniques show that $C_{B_2}(T_k^n) \geq 2n + \min(k, n-k+1) - 3$ for any $k \in [2:n-1]$.

In this chapter, we present new lower bounds for the monotone circuit complexity of the threshold functions. For arbitrary threshold functions, we develop a general theorem using an approach similar to Stockmeyer, Paul, and others which shows that

$$MC(T_k^n) \geq 2n + 2\min(k, n-k+1) - O(1)$$

for any $k \in [2:n-1]$. In particular, for the "majority function" $T_{\lceil n/2 \rceil}^n$, we show that

$$MC(T^n_{\lceil n/2 \rceil}) \geq 3n-7.$$

Using different techniques, we also study lower bounds on the monotone complexities for the function T^n_2. We present proofs of two results due to F. F. Yao[†], namely that

$$MC_{\vee}(T^n_2) = 2n-4 \text{ and } MC_{\wedge}(T^n_2) = \lceil \log n \rceil .$$

We then show that these lower bounds are not achievable simultaneously by proving that, for n a power of 2, any monotone circuit for T^n_2 with exactly $\lceil \log n \rceil$ $\vee$-gates must have at least $2n + 4 \log n - 9$ $\vee$-gates.

Before proceeding to proofs of the lower bounds, we first make some remarks on upper bounds for the M-circuit complexity of the threshold functions. As of the present, no linear (in k and n) upper bound exists on $MC(T^n_k)$ for arbitrary threshold functions. Hence the gap between the lower bounds mentioned above and the best known circuits are disappointingly large. For fixed values of k, L. Adleman has developed a scheme for constructing an M-circuit for T^n_k which has $kn + o(n)$ gates.[††]

[†] Private communication, 1976.

[††] Private communication, 1976.

In particular, he shows that

$$MC(T_2^n) = 2n + O(\sqrt{n})$$

and thus that the high order term of Yao's lower bound is exact[†] . While Adleman's method yields the best known circuits for small fixed values of n, a result due to A. Yao and F.F. Yao on building selection networks from comparators [Yao 1974] can be used to show that, for fixed k,

$$MC(T_k^n) \leq \lceil \log k+1 \rceil n + o(n).$$

Finally, if k is allowed to vary with n (for example if $k = \lceil n/2 \rceil$), the smallest known M-circuit which computes T_k^n has $O(n \log^2 n)$ gates. This bound is obtained by using an M-circuit of this size to sort the inputs in increasing order (Batcher in [Knuth 1973 pp. 111-114]) and then observing that $T_k^n(x_1,\ldots,x_n) = 1$ iff the k^{th} largest input is 1.

[†] We remark that in addition, Adleman (private communication, 1976) has proved a lower bound of $2n + \Omega(\sqrt{n})$ gates for any M-circuit which computes T_2^n, has exactly $\log n$ $\wedge$-gates, and is synchronous (for definition, see [Savage 1976],p.124).

Section A. The Class of all Threshold Functions

In this section we show that the general lower bound techniques used by numerous researchers [Stockmeyer 1977, Paul 1977, Schnorr 1974, Khasin 1970] for circuit complexity can yield a larger bound when the basis of allowable gates is restricted to M. The basic method of attack is as follows: To prove a lower bound on the combinational complexity of a particular Boolean function f, one attempts to show that there is a set of variables $\{x_i \mid i \in A \subseteq [1:n]\}$ and corresponding constants $(c_i \mid i \in A,\ c_i \in \{0,1\})$ such that the restricted function $f\,|_{x_i = c_i \text{ for } i \in A}$ is a function whose complexity is known to be bounded below by some quantity. If, in addition, one can show that in any minimal B-circuit or formula which computes f, the setting $x_i = c_i$ for $i \in A$ allows one to eliminate (as in Chapter 2) k gates because these gates now have at least one constant input, we then know that

$$C_B(f) \geq k + C_B(f\,|_{x_i = c_i \text{ for } i \in A});$$

hence a lower bound on $C_B(f)$ can be obtained. This is the general approach we take; in particular, a lower bound on the resticted function $f\,|_{x_i = c_i}$ will be known by induction.

Before describing the inductive hypotheses needed to prove our result, we first informally describe the method. The largest lower bound obtained from this method is for the majority function $T^n_{\lceil n/2 \rceil}$. We will reduce an M-circuit N for an arbitrary threshold function T^n_k to one for $T^{n-1}_{k'}$ by setting the input x_n to a constant. The setting $x_n = 0$ results in a circuit for T^{n-1}_k while the setting $x_n = 1$ results in a circuit for T^{n-1}_{k-1} . If the constant can be chosen arbitrarily and still allow a large number of gates to be eliminated, then the constant will be chosen so that k' is as close to $\lceil (n-1)/2 \rceil$ as possible.

Some notation will be useful.

Definition: The parity function, $p: \mathbb{N} \to \{0,1\}$, is defined by

$$p(n) = \begin{cases} 1 & \text{if } n \text{ is odd} \\ 0 & \text{if } n \text{ is even} \end{cases} .$$

The partial function $\ell: \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is defined by $\ell(n,k) = |n+1-2k|$ whenever $k \le n$.

The sign function, denoted $\text{Sign}: Z \to \{0,1\}$ is defined by

$$\text{Sign}(x) = \begin{cases} 1 & \text{if } x \ge 0 \\ 0 & \text{if } x < 0 . \end{cases}$$

The function S is defined by $S(n,k) = \mathrm{Sign}(n+1-2k)$ for $n,k\in \mathbb{N}$.

Note that $\ell(n,k)$ is a measure of the proximity of k to $(n+1)/2$. Observe that $p(n-\ell(n,k)) = 1$, and that for fixed n, the solution in $\mathbb{N}$ to the equation $\ell(k,n) = r$ is non-existent unless $r \leq n-1$ and $p(n-r) = 1$, in which case $k = (n+1-r)/2$ and $k = (n+1+r)/2$ are the only solutions.

<u>Definition</u>: Suppose that $k,\ell\in\mathbb{N}$, $\ell \leq n-3$, and $p(n-\ell) = 1$. Define $T_{n,\ell}$ = {threshold functions $T^n_k \mid \ell(n,k) = \ell$}.

For completeness, we define $T_{n,\ell} = \emptyset$ if $\ell > n-3$ or $p(n-\ell) = 0$.

Define $MC(n,\ell) = \max\{MC(f) \mid f\in T_{n,\ell}\}$.

We observe that by the above note, if $\ell \leq n-3$ and $n-\ell$ is odd, then $T_{n,\ell} = \{T^n_{(n+1-\ell)/2}, T^n_{(n+1+\ell)/2}\}$, and the two members of $T_{n,\ell}$ are duals of each others.

The prinicpal result we prove is the following:

<u>5.1 Theorem</u>: Suppose $n,\ell\in\mathbb{N}$, $p(n-\ell) = 1$, and $\ell \leq n-3$ (so $n \geq 3$). Then $MC(n,\ell) \geq 3n-\max(\ell,1) - 6$.

Before proving Theorem 5.1, we observe the following corollaries:

<u>5.2 Corollary</u>: Suppose $n \geq 3$. Then $MC(T^n_{\lceil n/2 \rceil}) \geq 3n - 7$.

<u>Proof</u>: $T^n_{\lceil n/2 \rceil} \in \mathcal{T}_{n,p(n+1)}$. □ Corollary 5.2

<u>5.3 Corollary</u>: Suppose $n \geq 3$, $k \in [2:n-1]$, $k \neq (n+1)/2$. Then $MC(T^n_k) \geq 2n + 2(\min(k,n-k+1)) - 7$.

<u>Proof</u>: Assume $k \in [2:n-1]$ and $k \neq (n+1)/2$. Since $\ell(n,k) \leq n-3$ and $p(n-\ell(n,k)) = 1$, we have $T^n_k \in \mathcal{T}^n_{\ell(n,k)}$. But then, since $k \neq (n+1)/2$, $\ell(n,k) = |n+1-2k| = \max(n+1-2k, 2k-n-1) \neq 0$ and thus Theorem 5.1 implies

$$\begin{aligned} MC(T^n_k) &\geq 3n - \ell(n,k) - 6 \\ &= 3n - \max(n+1-2k, 2k-n-1) - 6 \\ &= 2n + 2\min(k,n-k+1) - 7. \end{aligned}$$

□ Corollary 5.3.

The property of classes $\mathcal{T}_{n,\ell}$ which we will exploit is the following:

<u>5.4 Lemma</u>: Suppose $n,\ell \in \mathbb{N}$, $n \geq 3$, and $\ell \leq n-3$. If $f \in \mathcal{T}_{n,\ell}$, then there is a constant $s \in \{0,1\}$ such that

(a) $f\,|_{x_n = s} \in \mathcal{T}_{n-1,\ell+1}$ if $\ell \leq n-5$

and

(b) $f\,|_{x_n = \neg s} \in \mathcal{T}_{n-1,|\ell-1|}$.

Proof: Suppose $f = T_k^n \in \mathcal{T}_{n,\ell}$. We claim that $s = S(n,k)$ satisfies the properties claimed by the lemma. This is proven by examining both of the possible members of $\mathcal{T}_{n,\ell}$. For example, if

$$f = T^n_{(n+1-\ell)/2} \in \mathcal{T}_{n,\ell},$$

then $S(n,\ell) = 1$, so

$$f\,|_{x_n = 1} = T^{n-1}_{(n-1-\ell)/2} \in \mathcal{T}_{n-1,\ell+1} .$$

The other case is similar. □ Lemma 5.4.

We will use one additional lemma which gives information about the structure of B_2-circuits (or M-circuits) computing threshold functions.

5.5 Lemma [Schnorr 1974]: Let Ω be the basis B_2 or M, and let N be a minimal Ω-circuit for T_k^n for $n \geq 3$ and $k \in [2:n-1]$. Then there exists a gate $G \in \text{Gates}(N)$ and indices $i,j \in [1:n]$ such that

(1) $\text{Pred}(G) = \{x_i, x_j\}$

(2) $i \neq j$

and (3) outdeg $(x_j) \geq 2$[†].

Proof Sketch (For complete proof see [Schnorr 1974] or [Stockmeyer 1977]): Since N is an acyclic graph, one can clearly find a node G and indices i and j satisfying property (1). Since N is minimal, $i \neq j$. We now prove that one of x_i and x_j must have outdegree of at least 2. Suppose to the contrary that G is the unique gate in $\text{Succ}(x_i) \cup \text{Succ}(x_i)$, and suppose $\text{Res}(G,N) = g(x_i, x_j)$. Then two of the values $g(0,0)$, $g(0,1)$, and $g(1,1)$ must be equal since they are all members of $\{0,1\}$.

[†] Actually, Lemma 5.5 holds for any function $f \in B_n$ with the property that, for any pair of distinct inputs x_i and x_j,

$$\left| \left\{ f \Big|_{\substack{x_i = c_1 \\ x_j = c_2}} \;\middle|\; c_1, c_2 \in \{0,1\} \right\} \right| \geq 3.$$

Such a function is said to satisfy the "2-3 property". Lower bounds on the complexity of functions with this and more generalized properties have been previously studied [Savage 1976].

Consequently two of the functions $T_k^n \,|_{\substack{x_i = 0 \\ x_j = 0}}$, $T_k^n \,|_{\substack{x_i = 0 \\ x_j = 1}}$,

and $T_k^n \,|_{\substack{x_i = 1 \\ x_j = 1}}$ must be equal. Since these three functions are distinct, this is a contradiction, so one of x_i or x_j must have outdegree at least 2. □ Lemma 5.5.

We can now return to the proof of Theorem 5.1.

Proof of Theorem 5.1: The proof is broken down into two stages:

(i) the result is first proven for $\ell = n-3$ for all n

and

(ii) the result is proven for $\ell \le n-5$ by induction on n, using as basis part (i).

Part (i): Suppose $\ell = n-3$. In this case, $T_{n,\ell} = \{T_2^n, T_{n-1}^n\}$. Lower bounds of $2n-3$ on C_{B_2}, and hence MC, for both of these functions have been proven previously by Schnorr [1974]. This proof is repeated in [Stockmeyer 1977]. In fact, the lower bound of $2n-3$ on $MC(T_2^n)$ follows from Theorems 5.6 and 5.9, which will be proven later in this chapter independently of these results. The lower bound on $MC(T_{n-1}^n)$ follows by duality.

Part (ii): Suppose that $\ell \leq n-5$, $f \epsilon T_{n,\ell}$, and N is a minimal M-circuit which computes f. We break the proof up into several cases.

Case 1: There is an input x_i with outdeg $(x_i) \geq 3$. See Figure 5.1.

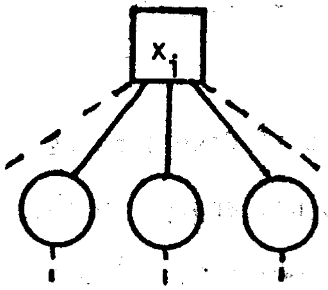


Fig. 5.1 Case 1

W.l.o.g., by renumbering the input nodes of N we assume that $i = n$. In this case, by Lemma 5.4, there is a constant $s \epsilon \{0,1\}$ such that $f' = f|_{x_n = \neg s} \epsilon T_{n-1,|\ell-1|}$. By setting x_n to $\neg s$ in N, all gates in Succ(x_n, N) may be eliminated, so the resulting contracted network N′ has at least 3 fewer gates than N. Hence $MC(f) = C_M(N) \geq C_M(N') + 3$

$$\geq MC(f') + 3.$$

If $\ell \geq 1$, then $f' \epsilon T_{n-1,\ell-1}$, so by induction

$$\begin{aligned} MC(f') + 3 &\geq 3(n-1) - \max(\ell-1,1) - 6 + 3 \\ &= 3n - \max(\ell-1,1) - 6 \\ &\geq 3n - \max(\ell,1) - 6. \end{aligned}$$

On the other hand,

if $\ell = 0$, then $f' \in T_{n-1,1}$, so

$$\begin{aligned} MC(f') + 3 &\geq 3(n-1) - \max(1,1) - 6 + 3 \\ &= 3n-7 = 3n - \max(\ell,1) - 6 \end{aligned}$$

This proves the result if Case 1 holds. Assume in the remainder that Case 1 does not apply. Assume in addition that x_i, x_j, and G are as in Lemma 5.5; w.l.o.g., we assume that $j = n$. Since Case 1 does not apply, outdeg $(x_n) = 2$, so we may let $\mathrm{Succ}(x_n) = \{G,H\}$.

<u>Case 2</u>: G is an $\wedge$-gate, and H is an $\vee$-gate. See Figure 5.2.

In this case neither G nor H is the output gate since $\mathrm{Res}(H,N)\,|_{x_n = 1}$ is constant while $f\,|_{x_n = 1}$ is not; similarly $\mathrm{Res}(G,N)\,|_{x_n = 0} \neq f\,|_{x_n = 0}$. Hence we can assume that A and B

are arbitrary members of Succ(G) and Succ(H) respectively. (Note that we are not assuming that A ≠ B.) See Figure 5.3.

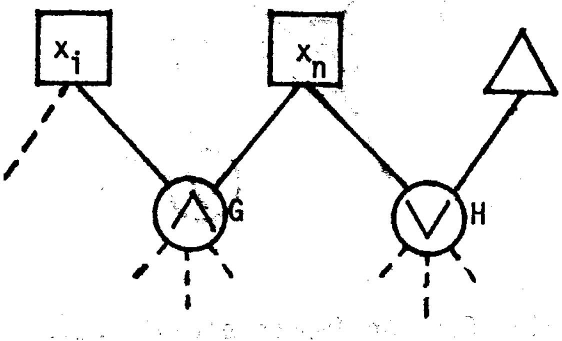


Fig. 5.2 Case 2

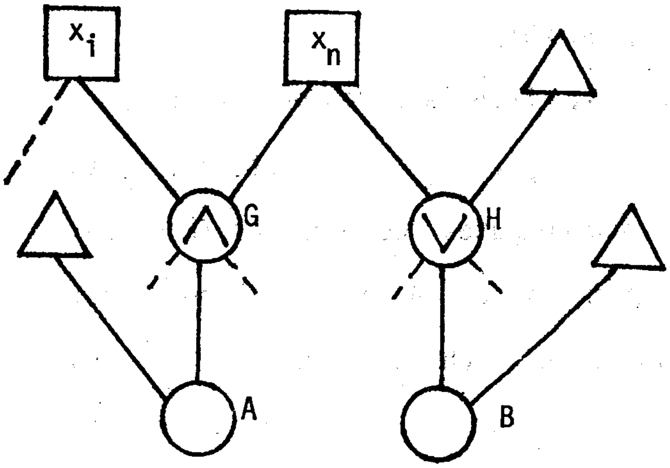


Fig. 5.3 Two Gates

We claim that $A \neq H$; that is, $H \notin Succ(G)$. If it were (see Figure 5.4) then $Res(H,N) = x_n \vee (x_n \wedge x_i) = x_n$ and the circuit

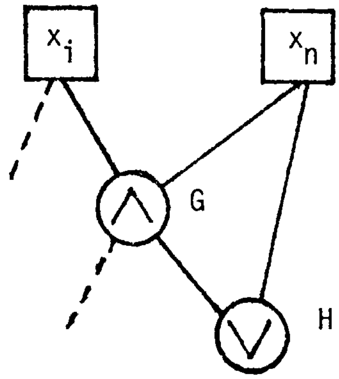


Fig. 5.4 An Impossible Situation

N is not minimal. Since $Pred(G) = \{x_i, x_n\}$, $G \notin Succ(H)$ and thus the gate(s) A and B are distinct from G and H. (In fact $A \neq B$ but we do not need this).

By Lemma 5.4, there is some constant s such that the restriction $f' = f |_{x_n} = \neg s \in T_{n-1,|\ell-1|}$. By setting x_n to $\neg s$ in N, one of G or H now computes the constant function $\neg s$, and hence at least one input to either A or B is now constant. Hence the resulting network N′, which computes f′, can have at least 3 gates (G, H, and one of A or B) eliminated from it. As in Case 1, this implies that $MC(f) \geq 3n - \max(\ell,1) - 6$.

<u>Case 3</u>: G is an $\vee$-gate, and H is an $\wedge$-gate.

This case is handled similarly to Case 2.

Case 4: Both G and H are $\wedge$-gates. See Figure 5.5 .

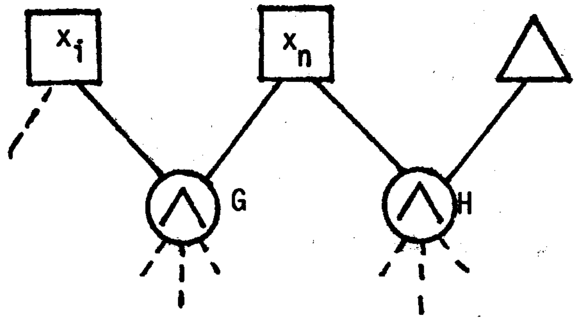


Fig. 5.5 Case 4

In this case, the objective will be to show that by setting x_n to the constant 0, at least 4 gates can be eliminated from N. Observe that by Lemma 5.4 , $f|_{x_n=0} \in T_{n-1,\ell+1} \cup T_{n-1,|\ell-1|}$.

As in Case 2, it is clear that neither G nor H is the output gate since $f|_{x_n=0}$ is not a constant function. Hence outdeg (G) ≥ 1 and outdeg (H) ≥ 1. Also $H \notin \text{Succ}(G)$ since if this were so then $\text{Res}(H,N) = x_n \wedge (x_n \wedge x_i) = x_n \wedge x_i$ (see Figure 5.6). But this means that two gates, G and H, compute the same function $x_n \wedge x_i$, a situation that never happens in a minimal circuit. In a similar fashion, it is possible to show that $G \notin \text{Succ}(H)$.

We now show that there are at least 2 gates in $\text{Succ}(G) \cup \text{Succ}(H)$. Suppose to the contrary that there is only 1 gate A in

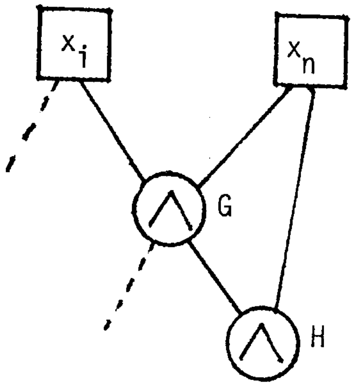


Fig. 5.6 Another Impossible Situation

Succ(G) ∪ Succ(H). See Figure 5.7. Let Pred(H) = { x_n,J } , and

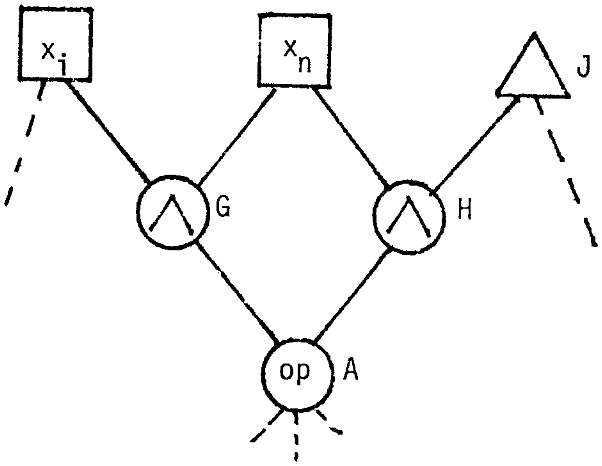


Fig. 5.7

suppose that A is an op-gate, where $op \in \{\wedge, \vee\}$. By using the identity (for any m.b.f. f)

$$(x_n \wedge x_i) \text{ op } (x_n \wedge f) = x_n \wedge (x_i \text{ op } f)$$

we may obtain a smaller circuit by replacing the subcircuit in Figure 5.7 by the following one with one fewer gate (Figure 5.8);

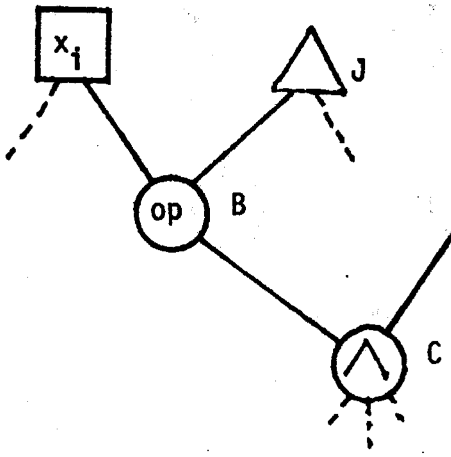


Fig. 5.8 The Rearrangement

that is, eliminate gates G, H, and A; add gates B and C as above and replace A by C in Pred(K) for every gate K in Succ(A,N). Thus, if N were minimal, there are at least two gates in Succ(G) ∪ Succ(H) (see Figure 5.9).

We now are in a position to show that at least 4 gates may be eliminated by setting $x_n = 0$. Let N′ be the circuit obtained by this evaluation, and note that Res(G,N′) = Res(H,N′) = 0. Hence all gates in Succ(G) ∪ Succ(H) have at least one input constant. Since this is at least 2 gates in addition to G and H, this means that at least 4 gates in N′ have a constant input and may be eliminated, yielding a reduced circuit N´´.

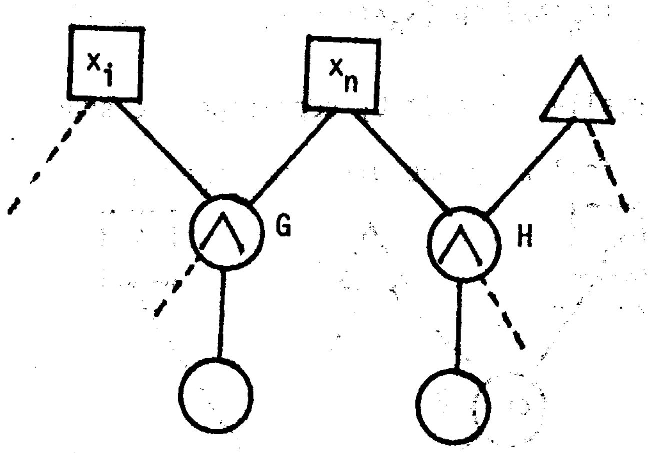


Fig. 5.9 Two More Gates

Finally, we show that this elimination of 4 gates yields the desired result. As mentioned $f' = f|_{x_n} = 0 \in T_{n-1,\ell+1} \cup T_{n-1,|\ell-1|}$. Since N´´ computes f´, we obtain

$$\begin{aligned} MC(f) &= \hat{C}_M(N) \geq 4 + C_M(N'') \\ &\geq 4 + MC(f') \\ &\geq 4 + \min(MC(n-1,\ell+1), MC(n-1,|\ell-1|) \\ &\qquad\qquad (\text{since } f' \in T_{n-1,\ell+1} \cup T_{n-1,|\ell-1|}) \end{aligned}$$

$$\geq 4 + \min(3(n-1) - \max(\ell+1,1)-6,\ 3(n-1) - \max(|\ell-1|,1)-6)$$
(by induction)

$$= 4 + 3(n-1) - \max(\ell+1,|\ell-1|,1) - 6$$

$$= 3n-\ell-6 \geq 3n - \max(\ell,1) - 6$$

and the proof is complete for Case 4.

Case 5: Both G and H are ∨-gates.

This case is handled dually to Case 4.

□ Theorem 5.1

Section B. The Monotone Circuit Complexity of Threshold 2

We now consider a specific set of threshold functions, T_2^n for $n \geq 2$, and demonstrate some known lower bounds on the monotone circuit complexity for this set of functions. We first obtain lower bounds on the number of ∨-gates and ∧-gates separately. The general technique used in previous sections of setting certain inputs to constants and eliminating gates will be used here as well. For the specific function T_2 we will be able not only to establish that a certain number of gates can be eliminated, but also to determine what types of gates can be eliminated.

5.6. Theorem: (F.F. Yao): If $n \geq 2$ then $MC_v(T_2^n) \geq 2n-4$.

Proof (Bloniarz): By induction on n. The case $n = 2$ is obvious.

For larger values of n, we use an argument similar to that used previously. Assume the statement of the theorem true for $n-1$, and suppose N is an M-circuit computing $T_2^n(x_1,\dots,x_n)$ with exactly the minimal number $MC_v(T_2^n)$ of v-gates. Assume further that N has the minimal total number of gates among all such circuits with $MC_v(T_2^n)$ v-gates. We show there is some variable, which by symmetry we assume is x_n, such that setting x_n to 0 results in a circuit from which at least two v-gates may be elinated. If the resulting contracted circuit is N', then N' computes $T_2^{n-1}(x_1,\dots,x_{n-1})$, and we obtain the inequality

$$\begin{aligned} MC_v(T_2^n) = \hat{MC}_v(N) &\geq \hat{MC}_v(N') + 2 \\ &\geq MC_v(T_2^{n-1}) + 2 \quad \text{(since } N' \text{ computes } T_2^{n-1}\text{)} \end{aligned}$$

and the inductive assumption completes the proof.

We use Lemma 5.5 to establish the result; w.l.o.g. we may assume that there is a gate G in N and inputs x_i and x_n to N

such that $i \neq n$, $\mathrm{Pred}(G) = \{x_i, x_n\}$ and $\mathrm{outdeg}\,(x_n) \geq 2$. See Figure 5.10. Clearly by setting x_n to 0 at least two gates may be eliminated. We prove that at least two of these gates are v-gates, but first introduce some useful notation.

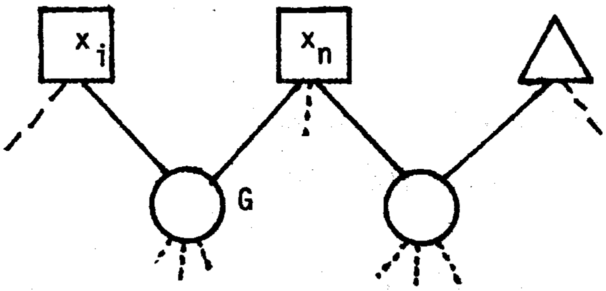


Fig. 5.10

Suppose that G and H are nodes of N. Recall that a <u>path</u> from G to H is a sequence of nodes $G_0, G_1, \ldots, G_k$ (for $k \geq 0$) of N such that $G_0 = G$, $G_k = H$ and $G_{i+1} \in \mathrm{Succ}(G_i)$ for $i \in [0:k-1]$.

Definition: An $\wedge$-chain from G to H is a path $G_0,\ldots,G_k$ such that G_i is an $\wedge$-gate for all $i\in[1:k]$. That is, an $\wedge$-chain is a path in the graph of N all of whose nodes, except possibly the first, are $\wedge$-gates. Note that a path consisting of a single node is an $\wedge$-chain regardless of the type of node.

Define

$$T(G,N) = \{H\in \mathrm{Succ}^*(G,N) \mid \text{there is an } \wedge\text{-chain from } G \text{ to } H\}$$

and

$$V(G,N) = \{H\in \mathrm{Succ}^*(G,N) \mid H \text{ is an } \vee\text{-gate and } \mathrm{Pred}(H)\cap T(G)\neq\emptyset\}\ .$$

(where again we omit mention of N if there is no ambiguity).

The importance of these definitions is as follows.

5.7 Lemma: Suppose N is an arbitrary M-circuit, and G and H are nodes in N. Suppose further that $\mathrm{Res}(G,N) = \underline{0}$. Then

(a) If $H\in T(G)$, $\mathrm{Res}(H,N) = \underline{0}$;

(b) If $H\in V(G)$, then H has at least one constant input and may be eliminated from N;

and

(c) If $H\in \mathrm{Succ}^*(G)$ and $\mathrm{Res}(H,N) \neq 0$, then every path from G to H contains a node in $V(G)$.

Proof: (a) and (b) are obvious. To prove (c), suppose $\text{Res}(H,N) \neq \underline{0}$. Then, by (a), there is no $\wedge$-chain from G to H. Thus every path from G to H contains at least one $\vee$-gate, and the first such is easily seen to be a member of $V(G)$.

□ Lemma 5.7.

We proceed with the proof of Theorem 5.6. Suppose U is the output gate of N, and let N be the circuit obtained when the constant $\underline{0}$ is substituted for x_n. By Lemma 5.7(b), this setting $x_n = 0$ allows us to eliminate all the $\vee$-gates in $V(x_n,N)$ from N'; if $|V(x_n,N)| \geq 2$, then the proof is complete†. We therefore need consider only $|V(x_n)| < 2$.

Because T_2^n depends on x_n, we know that $U \in \text{Succ}^*(x_n,N)$. Since $\text{Res}(U,N')$ is not the constant function $\underline{0}$, we know that there is at least one gate in $V(x_n)$ by Lemma 5.7(c).

† Note that T and V are defined in terms of the structural properties of the circuit; in fact $V(x_n,N) = V(\text{ZERO},N')$, and $T(x_n,N) = T(\text{ZERO},N')$. We refer to these sets of gates as $V(x_n)$ and $T(x_n)$ respectively.

So we may assume that $V(x_n)$ contains exactly one gate H. Note that $U \epsilon Succ^*(H,N)$ since N was an $\vee$-minimal circuit for T_2^n. Suppose that $Res(H,N') = \underline{0}$. Again using Lemma 5.7(c), we know that $V(H,N') \neq \emptyset$. Since the graph of N is acyclic, it is impossible for H to be a member of V(H), so $|\{H\} \cup V(H)| \geq 2$. By Lemma 5.7(b), all nodes in $\{H\} \cup V(H)$ may be eliminated from N′, constituting at least two $\vee$-gates.

We may conclude the proof by demonstrating that the remaining case, namely when $V(x_n) = \{H\}$ and $Res(H,N') \neq \underline{0}$, cannot happen. Since H is a gate, we may assume that $Pred(H) = \{I,J\}$ (where it is possible that G is one of I,J, or H). Since $H \epsilon V(x_n)$, at least one of I or J must be in $T(x_n)$; we assume w.l.o.g. that $I \epsilon T(x_n)$. The steps of the proof that this case cannot occur are embodied in the following lemma.

<u>5.8 Lemma</u>: Suppose G, H, I, and J are as above. Then

(a) Every path from x_n to U in N must pass through H;

(b) J does not depend structurally on x_n in N;

(c) G is an $\wedge$-gate;

and

(d) $I \epsilon T(x_i)$.

<u>Proof of Lemma</u>: Result (a) follows from the fact that $Res(U,N') \neq \underline{0}$. By Lemma 5.7(c), we then know that every path from ZERO to U in N′ must pass through H; that is, every path from

x_n to U in N must pass through H.

Since $I \in T(x_n)$, we know $\mathrm{Res}(I,N') = \underline{0}$ by Lemma 5.7(a). Hence $\mathrm{Res}(J,N') \neq \underline{0}$ since $\mathrm{Res}(I,N') \vee \mathrm{Res}(J,N') = \mathrm{Res}(H,N') \neq \underline{0}$. Thus by Lemma 5.7(c), any path from x_n to J would contain H, the unique gate in $V(x_n)$. Since $J \in \mathrm{Pred}(H)$ by definition, and N is acyclic, we conclude that there is no path from x_n to J. Hence (b) is established.

If G were an $\vee$-gate, then G would be equal to H and there would be exactly one path consisting of a single arc from x_n to H. Since outdeg $(x_n) \geq 2$, there is a gate $G' \neq G$ in $\mathrm{Succ}(x_n)$. Since there is a path from G' to U in N', there must be a path from G' to H in N. Since both members of $\mathrm{Pred}(G,N)$ are input nodes, this is impossible, so (c) is established.

Finally, since G is an $\wedge$-gate, $V(G) \subseteq V(x_n)$. By Lemma 5.7(c), we know that $V(G) \neq \emptyset$ since there is a chain from G to U in N' and since $\mathrm{Res}(G,N') = \underline{0}$. Hence $H \in V(G)$, so by definition at least one of I or J is a member of $T(G)$. If $J \in T(G)$, then $J \in T(x_n)$ in contradiction to Lemma 5.8(b); hence $I \in T(G)$. Since $T(G) \subseteq T(x_i)$, the proof of (d) is complete.

□ Lemma 5.8.

Now let N'' be the circuit obtained from N (<u>not</u> N') by setting x_i to 0. $\mathrm{Res}(U,N'')$, being the threshold 2 function of the remaining n-1 variables $\{x_1,\ldots,x_n\} - \{x_i\}$, depends on x_n. By Lemma 2.9, we know that there is a path C in N'' from x_n

to U such that every gate in C depends functionally on x_n. Since H is on every path from x_n to U in N by Lemma 5.8(a), it must appear on every path in N´´ and hence on C. Thus Res(H,N´´) depends on x_n in N´´.

However, by Lemmas 5.7(a) and 5.8(d), Res(I,N´´) = $\underline{0}$, so Res(I,N´´) does not depend functionally on x_n. Similarly, by Lemma 5.8(b), J cannot depend functionally on x_n in N, and hence in N´´, so Res(J,N´´) does not depend functionally on x_n. Thus Res(H,N´´) does not depend on x_n, a contradiction.

□ Theorem 5.6.

We can also get a lower bound on the number of $\wedge$-gates in any M-circuit which computes T_2^n. Here the technique used is similar, but we must set more than one input to a constant to eliminate a single $\wedge$-gate.

<u>5.9 Theorem</u>: (F.F. Yao): Suppose $n \geq 1$. Then $MC_\wedge(T_2^n) \geq \lceil \log_2 n \rceil$.

<u>Proof</u>: We prove this fact by strong induction on n. For n = 1 the result is trivial.

Now suppose $n \geq 2$ and the statement of the theorem is true for all $n' < n$. Suppose N is an $\wedge$-minimal M-circuit which computes T_2^n. Since $n \geq 2$ and since $x_i \notin PI(T_2^n)$ for any index $i \in [1:n]$, N must contain at least one $\wedge$-gate. Let G be a top-most $\wedge$-gate in N; that is, G is an $\wedge$-gate such that every gate in $\text{Pred}^+(G)$ is not an $\wedge$-gate. Clearly such a gate exists since N is acyclic. Suppose $\text{Pred}(G) = \{H,J\}$. See Figure 5.11.

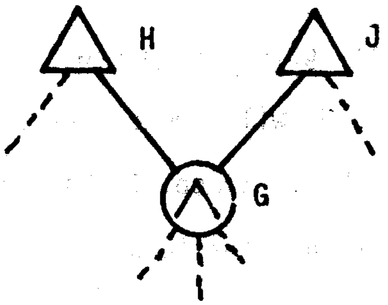


Fig. 5.11 A Top-most $\wedge$-Gate

Since G is a top-most $\wedge$-gate, all gates in $\text{Pred}^+(G)$ are either $\vee$-gates or inputs. Hence $\text{Res}(H,N) = \bigvee_{i \in A} x_i$ and $\text{Res}(J,N) = \bigvee_{i \in B} x_i$ for some non-empty subsets A and B of $[1:n]$. Observe that

$$\text{Res}(G,N) = \Big(\bigvee_{i \in A \cap B} x_i\Big) \wedge \Big(\bigvee_{\substack{i \in A-B \\ j \in B-A}} (x_i \wedge x_j)\Big) \qquad (1)$$

Let C denote the smaller of the sets $A-B$ and $B-A$ and let c denote $|C|$. Then one can easily check that $c \leq n/2$. By renumbering the inputs, we may assume w.l.o.g. that $C = [n-c+1:n]$.

Let N' denote the circuit which results when the entire set of variables $\{x_i \mid i \in C\}$ is set to 0. Note that N' computes the function $T_2^n\,|_{x_i = 0 \text{ for } i \in C} = T_2^{n-c}$. By (1) we know that $\mathrm{Res}(G;N') = \bigvee_{i \in A \cap B} x_i$.

We now modify N' to eliminate gate G. Construct a new tree of $\vee$-gates and input nodes which computes the function $\bigvee_{i \in A \cap B} x_i$ at a gate G'. Eliminate G from N', and replace G by G' in $\mathrm{Pred}(K)$ for all gates $K \in \mathrm{Succ}(G,N')$. (If G were the output gate of N, then G' will now be the output gate.) Call the resulting circuit N''. Since G has been replaced in N'' by a node which computes the same function, N'' also computes T_2^{n-c} . Since N'' has one fewer $\wedge$-gate than N and since N had a minimal number of $\wedge$-gates computing T_2^n, we know that N'' does not compute T_2^n and thus $c \geq 1$. Hence

$$\begin{aligned} MC_\wedge(T_2^n) = \hat{MC}_\wedge(N) &= 1 + \hat{MC}_\wedge(N'') \\ &\geq 1 + MC_\wedge(T_2^{n-c}) \\ &\geq 1 + \log(n-c) \text{ (by induction)} \\ &\geq 1 + \log(n/2) \text{ (since } c \leq n/2) \\ &= \log(n) \end{aligned}$$

and the theorem is proved.

□ Theorem 5.9.

In preparation for future results in the chapter, we note the following additional facts gleaned from the proof of Theorem 5.9. Suppose n is an exact power of 2, and N is an M-circuit for T_2^n with exactly $\log n$ $\wedge$-gates (which we later show is possible). If G is a top-most $\wedge$-gate of N, then if c (as defined in the proof of Theorem 5.9) is less than $n/2$, the setting of the variables $\{x_i \mid i \in C\}$ to 0 leaves a circuit N' for a function $T_2^{n'}$, where $n' > n/2$, which has 1 fewer $\wedge$-gate than N. But this is impossible since then

$$\begin{aligned} \log n &= \hat{MC}(N) \\ &\geq \hat{MC}(N') + 1 \\ &\geq \log n' + 1 \\ &= \log (2n') \\ &> \log n \quad . \end{aligned}$$

Hence we have proved

<u>5.10 Corollary</u>: Suppose that $n = 2^r$ for $r \in \mathbb{N}$, and that N is an M-circuit for T_2^n with exactly $\log n$ $\wedge$-gates. If G is a top-most $\wedge$-gate of N and $\mathrm{Pred}(G) = \{H,J\}$, then there is a partition of $[1:n]$ into two sets A and B such that $|A| = |B| = n/2$ and such that $\mathrm{Res}(H) = \bigvee_{i \in A} x_i$ and $\mathrm{Res}(J) = \bigvee_{j \in B} x_j$.

We proceed with our discussion of T_2^n by demonstrating that each of these lower bounds is attainable in some M-circuit computing T_2^n.

An M-circuit with exactly $2n-4$ $\vee$-gates for T_2^n (for $n \geq 2$) is obtained by use of the following recursive constructions. We will show inductively that, for $n \geq 2$, there is a two-output M-circuit on inputs $\{x_1,\ldots,x_n\}$ containing $2n-3$ $\vee$-gates which computes $T_2^n(x_1,\ldots,x_n)$ and $T_1^n(x_1,\ldots,x_n)$. For $n = 2$, the circuit is given in Figure 5.12.

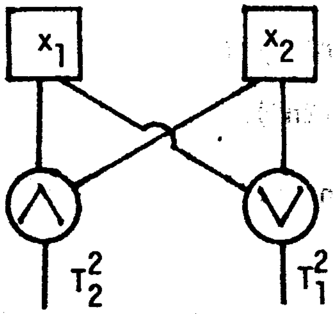


Fig. 5.12 A circuit for n=2

Assuming the fact true for n, we may construct such a circuit for n+1 variables with 2 additional $\vee$-gates (and an $\wedge$-gate) by use of the recurrence relations

$$T_2^{n+1}(x_1,x_2,\ldots,x_n,x_{n+1}) = T_2^n(x_1,\ldots,x_n) \vee (x_{n+1} \wedge T_1^n(x_1,\ldots,x_n)$$

and

$$T_1^{n+1}(x_1,\ldots,x_n,x_{n+1}) = x_{n+1} \vee T_1^n(x_1,\ldots,x_n).$$

See Figure 5.13.

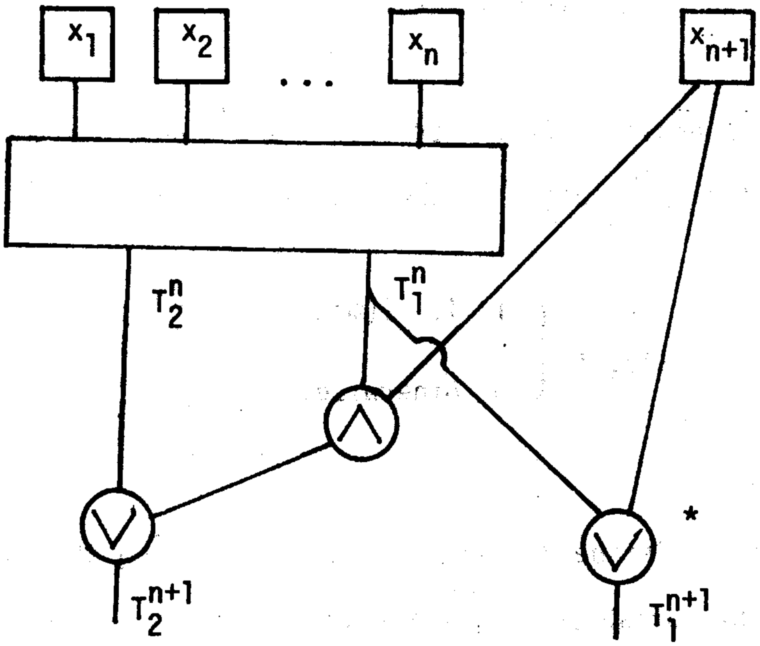


Fig. 5.13 The Recursive Construction

A circuit for T_2^n with exactly $2n-4$ $\vee$-gates may be thus obtained since at the last step in this construction of T_2^n, there is no need for the $\vee$-gate marked by $*$ in Figure 5.13.

5.11 Corollary: For $n \geq 2$, $MC(T_2^n) = 2n-4$.

As for exhibiting an M-circuit for T_2^n with exactly $\lceil \log n \rceil$ $\wedge$-gates, we first describe a general method for such a construction which was communicated to the author by F.F. Yao and A. Yao. For the remainder of this chapter it will be convenient to have the variable indices begin with 0 rather 1.

Definition: Suppose $\ell \in \mathbb{N}$. For $0 \leq \ell \leq 2^\ell$ and $1 \leq j \leq \ell$, define $(i)_j$ to be the j^{th} digit in the binary expansion of i, that is,

$$(i)_j = \begin{cases} 1 & \text{if } i(\text{mod } 2^j) \geq 2^{j-1} \\ 0 & \text{otherwise.} \end{cases}$$

Suppose $n \in \mathbb{N}$. Define for each $j \in [1:\lceil \log (n) \rceil]$ the following subsets of the variables $\{x_i \mid i \in [0:n-1]\}$.

$$A_j^n = \{x_i \mid i \in [0:n-1] \text{ and } (i)_j = 0\}$$

and $$B_j^n = \{x_i \mid i \in [0:n-1] \text{ and } (i)_j = 1\} .$$

Define corresponding Boolean functions $a_j^n, b_j^n \epsilon B_n$ by

$$a_j^n(x_0,\ldots,x_{n-1}) = \bigvee_{i \epsilon A_j} x_i$$

and $$b_j^n(x_0,\ldots,x_{n-1}) = \bigvee_{i \epsilon B_j} x_i \quad .$$

(We omit mention of n if it is clear from context.) Finally, the radix-join function $F^n \epsilon B_{n,\, 2\lceil \log n \rceil}$ is defined by

$$F^n = (a_1, b_1, a_2, b_2, \ldots, a_{\lceil \log n \rceil}, b_{\lceil \log n \rceil}).$$

An M-circuit which computes F^n is called a radix-join network.

It is easy to see the relationship between the radix-join function and the function "Threshold 2." In fact, with the definition given above.

$$T_2^n = \bigvee_{j=1}^{\lceil \log n \rceil} (a_j \wedge b_j). \qquad (2)$$

To prove (2), note that the sets of variables A_j and B_j are disjoint for each $j \epsilon [1 : \lceil \log n \rceil]$. Hence, letting h be the function defined by the right hand side of (2), we know that $h \leq T_2^n$.

Conversely, for any pair of variables x_k and x_ℓ, with $k \neq \ell$, there is some position j at which the binary representation of k and ℓ differ. Hence $x_k x_\ell \epsilon PI(a_j \wedge b_j)$, so $x_k x_\ell \leq h$. This fact being true for every $k \neq \ell$ in $[0:n-1]$, we know that $T_2^n \leq h$. See Figure 5.14.

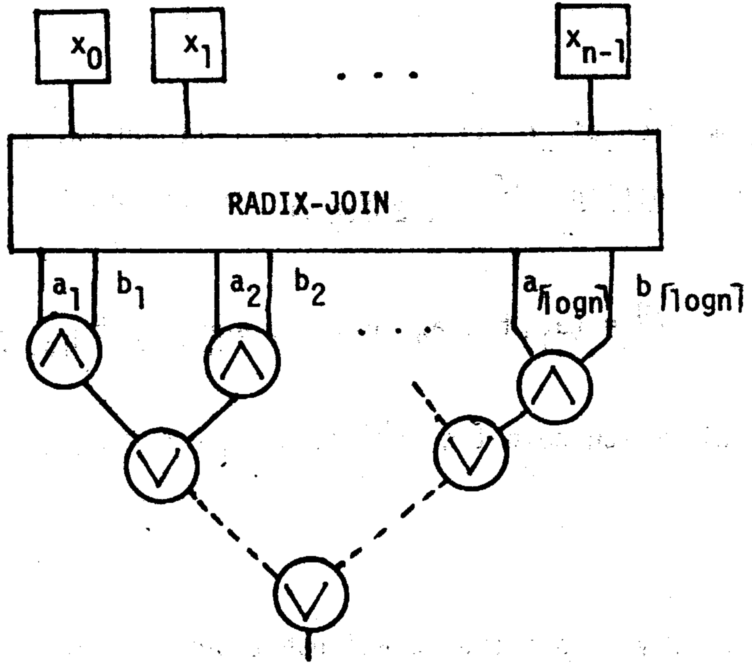


Fig. 5.14 An M-circuit for T_2^n

Applying the decomposition (2) to complexities, we have the following:

<u>5.12 Lemma</u>: Suppose $n \in \mathbb{N}$, and F^n is the radix-join function. Then

$$MC(T_2^n) \leq MC(F^n) + 2 \lceil \log n \rceil - 1$$

and $$MC_\wedge(T_2^n) \leq MC_\wedge(F^n) + \lceil \log n \rceil$$

Since F^n is defined only in terms of v-gates, $MC_\wedge(F^n) = 0$; hence we have:

<u>5.13 Corollary</u>: For $n \geq 1$, $MC_\wedge(T_2^n) = \lceil \log n \rceil$.

From the formula-definition of the radix-join function one can clearly construct a radix-join network with $O(n \log n)$ v-gates. A straightforward recursive construction yields a radix-join network with $3n - 2 \log n - 4$ v-gates, yielding an M-circuit for T_2^n using Lemma 5.12 with a total of $3n-5$ gates. Several substantially different M-circuits for T_2^n with this size can be constructed, including the circuit in Corollary 5.11. L. Adleman proved that asymptotically fewer than $3n-5$ gates are necessary[†], and using his techniques

[†] Personal communication, 1976

one can prove the following:

<u>5.14 Theorem</u>: Suppose $n \in \mathbb{N}$. Then

$$C_{\{v\}}(F^n) \leq 2n + O(\sqrt{n}).$$

<u>Proof</u>: Suppose $n > 4$ and let $\ell = \lceil \log n \rceil$. Let $n_1 = 2^{\lfloor \ell/2 \rfloor}$ and $n_2 = \lceil n/n_1 \rceil$. We will use the following recursive expansion to construct an efficient radix-join network for n variables. Let X denote the set of variables $\{x_i \mid i \in [0:n-1]\}$.

Suppose that $p \in [0:n_1-1]$ and $q \in [0:n_2-1]$. Define subsets of the variables X for such p and q by $L_p' = \{x_i \mid i \in [0:n-1], i \pmod{n_1} = p\}$ and $H_q' = \{x_i \mid i \in [0:n-1], \lfloor i/n_1 \rfloor = q\}$. Informally if all variables are thought to have indices whose binary representation is padded to length exactly ℓ, then L_p' is the set of variables which have a binary index whose lower order $\lfloor \ell/2 \rfloor$ bits are equal to the binary representation of p. Sinilarly, H_q' is the set of variables whose binary index has a high order $\lceil \ell/2 \rceil$ bits equal to the binary representation of q. We have corresponding functions L_p and H_q in B_n where L_p is the disjunction of all variables in L_p', and H_q is the disjunction of all variables in H_q'. Note that each variable $x_i \in X$ is a member of exactly one of the sets $\{L_p' \mid p \in [0:n_1-1]\}$, namely the unique p such that $i \pmod{n_1} = p$. Similarly, it is easy to verify that the collection $H_0, H_1, \ldots, H_{n_2-1}$ is a partition of the set X and that $x_i \in H_q$ where $q = \lfloor i/n_1 \rfloor$. Moreover, none of the sets H_q is empty for $q \in [0:n_2-1]$ since $q \cdot n_1 \leq n_1 n_2 - n_1 < n$ and $x_{q \cdot n_1} \in H_q$.

We claim that the relationship

$$F^n = (F^{n_1}(L_0, L_1, \ldots, L_{n_1-1}),\ F^{n_2}(H_0, \ldots, H_{n_2-1})) \qquad (3)$$

holds, where we use the obvious isomorphism between $\{0,1\}^{2\ell}$ and $\{0,1\}^{2(\lfloor \ell/2 \rfloor)} \times \{0,1\}^{2(\lceil \ell/2 \rceil)}$.

Assuming this for the moment, we describe an efficient radix-join network using (3). Using the definitions of H_p and L_q, we construct an independent $\{\vee\}$ - circuit (which happens to be fan-out free and is thus a formula) for each of the functions $L_0, L_1, \ldots, L_{n_1-1}$ and the functions $H_0, H_1, \ldots, H_{n_2-1}$. To count the number of $\vee$-gates which this takes, suppose $|L_p| = \ell_p$ for $p \in [0:n_1-1]$. Then, since L_p is a disjunction of single variables, $C_{\{\vee\}}(L_p) = \ell_p - 1$ since $\ell_p \neq 0$ for each p. Hence, the total number of $\vee$-gates required to compute all the functions $L_0, L_1, \ldots, L_{n_1-1}$ is

$$\sum_{p \in [0:n_1-1]} (\ell_p - 1) = \Big(\sum_{p \in [0:n_1-1]} \ell_p\Big) - n_1$$

$$= n - n_1$$

since each $x_i \epsilon X$ is a member of exactly one of the sets L_p' for some $p \epsilon [0:n_1-1]$. Similarly, one can compute all the functions H_q for $q \epsilon [0:n_2-1]$ with a total of $n - n_2$ v-gates, giving a total of $2n - n_1 - n_2$ gates in all.

We now use the n_1 functions L_p for $p \epsilon [0:n_1-1]$ as inputs to a recursively constructed radix-join network F^{n_1}, and likewise the n_2 functions H_q for $q \epsilon [0:n_2-1]$ as inputs to a recursively constructed radix-join network F^{n_2}. Using equation (3), this yields a circuit for F^n. (See Figure 5.15.)

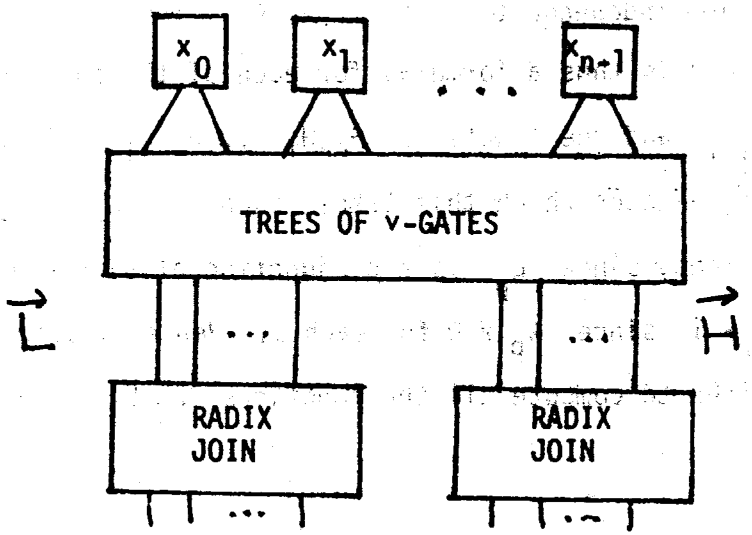


Fig. 5.15 The Recursive Radix-join Network

If we let $T(n) = C_{\{v\}}(F^n)$, then we have the recurrence relationship

$$T(n) \le 2n - n_1 - n_2 + T(n_1) + T(n_2).$$

Since $n_1 \le \sqrt{2n}$ and $n_2 \le \sqrt{2n}$ and since $T(4) = 4$, this recurrence can be bounded by

$$T(n) \le 2n + \underbrace{4(n/2)^{1/2} + 8(n/2)^{1/4} + 16(n/2)^{1/8} + \dots}_{(\log\log(n/2) \quad \text{terms})}$$

$$\le 2n + \sum_{k=1}^{\log\log(n/2)} 2^{k+1}(n/2)^{1/2^k}$$

which establishes the theorem since the summation is $O(\sqrt{n})$.

It remains to prove equation (3). Suppose that A_j^n, B_j^n, a_j^n, and b_j^n are as in the definition of F^n. Suppose that $j \le \lfloor \ell/2 \rfloor$. Then $A_j^n = \{x_i \mid (i)_j = 0\} = \bigcup_{(p)_j = 0} L'_p$ since $(i)_j = 0$ iff $(p)_j = 0$ where $i \pmod{n_1} = p$. Hence $a_j^n = \bigvee_{(i)_j = 0} L_i$, so a_j^n is the corresponding component $a_j^{n_1}$ of $F^{n_1}(L_0,\dots,L_{n_1-1})$. Similarly b_j^n is the component $b_j^{n_1}$ of $F^{n_1}(L_0,\dots,L_{n_1-1})$.

A similar proof demonstrates that when $\lfloor \ell/2 \rfloor < j \le \ell$, a_j^n is the component $a^{n_2}_{j-\lfloor \ell/2 \rfloor}$ of $F^{n_2}(H_0,\ldots,H_{n_2-1})$ and b^n_j is the component $b^{n_2}_{j-\lfloor \ell/2 \rfloor}$.

□ Theorem 5.14.

Applying Lemma 5.12, we have

<u>5.15 Corollary</u>: (Adleman) Suppose $n \in \mathbb{N}$. Then

$$MC(T^n_2) \le 2n + O(\sqrt{n}).$$

We will have more to say about this method in Section D.

<u>Section C - M-circuits for Threshold 2 which are $\wedge$-minimal</u>

In the previous section, we demonstrated lower bounds of $2n-4$ and $\lceil \log_2 n \rceil$ on the number of $\vee$-gates and $\wedge$-gates respectively necessary to compute T^n_2 by an M-circuit, and demonstrated circuits which achieved each of these bounds. The question naturally arises of whether there is a single M-circuit which computes T^n_2 and which simultaneously achieves this minimum number of $\wedge$-gates and $\vee$-gates, and in this section we show this is impossible for infinitely many values of n. Our technique will be to establish larger bounds on the number of $\vee$-gates necessary to compute T^n_2 (for n a power of 2) for any M-circuit which has exactly $\log n$ $\wedge$-gates.

We begin by examining the structure of such circuits with exactly $\log n$ $\wedge$-gates. In Section B, we saw that such a circuit may be constructed from a radix-join network. In this section, we will show that it is possible to extract a radix-join network from any M-circuit for T_2^n with exactly $\log n$ $\wedge$-gates. For now, we assume that $n = 2^r$ is an exact power of 2 (for some $r \geq 2$), and that N is an M-circuit which computes T_2^n, has exactly r $\wedge$-gates, and among all such circuits has a minimal number t of $\vee$-gates.

5.16 Theorem: Suppose n,N, and t are as above. Then an M-circuit R_n exists which computes F^n, the radix-join function, and which has $t - \log n + 1$ $\vee$-gates and no $\wedge$-gates.

Proof: We show how to extract R_n from N. We first show that there is a restructuring N_0 of N which also computes T_2^n in which there is only a single $\wedge$-gate on any path from an input node to the output gate - that is, there are no $\wedge$-gates in $\text{Pred}^+(G,N_0) \cup \text{Succ}^+(G,N_0)$ for any $\wedge$-gate G. Call such a circuit a single-level circuit. Moreover, we show that it is possible to construct N_0 with no additional $\wedge$-gates or $\vee$-gates.

Suppose that N does not have this property, and let U denote the (unique) output gate of N. Let G be a top-most $\wedge$-gate in N which has at least one $\wedge$-gate below it; that is, $\text{Pred}^+(G,N)$ contains no $\wedge$-gates and $\text{Succ}^+(G,N)$ contains at at least one $\wedge$-gate. Observe that $G \neq U$ by definition since $Succ^{+}(U,N) = \emptyset$. We show how to re-wire the circuit so that G is no longer on a path with any other $\wedge$-gate.

We first construct a circuit N' as follows. Add an additional $\vee$-gate U', and set $Pred(U',N') = \{G,U\}$. In addition , for every gate $H \in Succ(G,N)$, replace G in $Pred(H)$ by the constant node ZERO. See Figure 5.16.

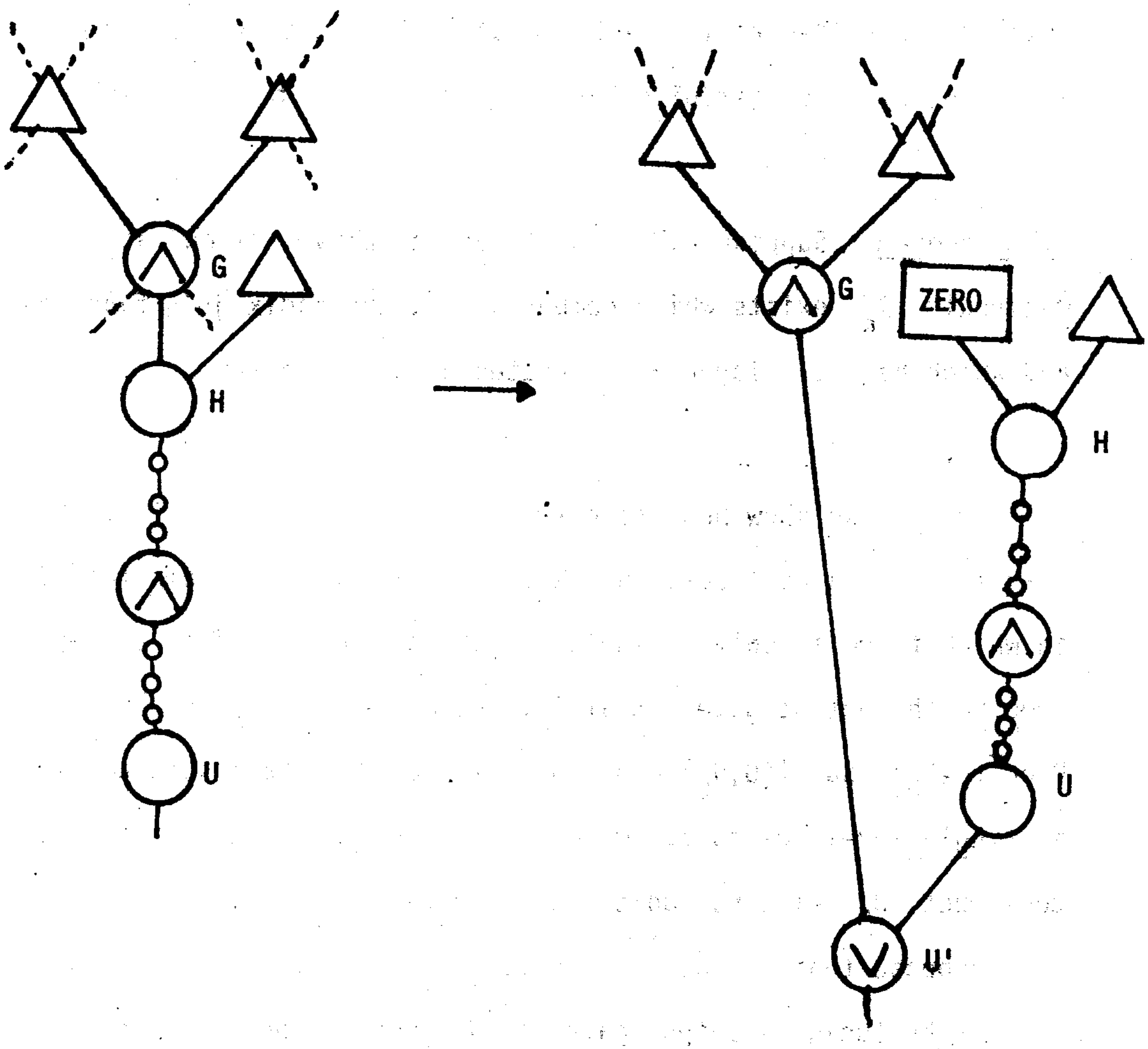


Fig. 5.16 The Reconstruction

Then simplify this circuit to yield a third circuit N'' by simplifying any gates with at least one input a constant function.

Since outdeg $(G,N) \geq 1$, this removes from N' all gates in $V(\mathrm{ZERO},N')$ (as defined in Section 5B) and hence at least 1 $\vee$-gate, so N' has at most t $\vee$-gates and at most r $\wedge$-gates.

We claim that $\mathrm{Res}(U',N'') = T_2^n$, and hence N'' has exactly t $\vee$-gates and r $\wedge$-gates. (Note that $\mathrm{Res}(U',N'') = \mathrm{Res}(U',N')$). To prove this, assume that $\vec{c} = (c_1,c_2,\ldots,c_n)$ is an arbitrary constant in $\{0,1\}^n$. We show that

$$\mathrm{Res}(U',N')(\vec{c}) = T_2^n(\vec{c}).$$

Since N had $\log n$ $\wedge$-gates, by Corollary 5.10 we know that there are subsets A and B of $[0:n-1]$ such that $A \cap B = \emptyset$ and $\mathrm{Res}(G,N) = \bigvee_{\substack{i \in A \\ j \in B}} x_i x_j$. Hence $\mathrm{Res}(G,N) \leq T_2^n$. Thus, in the case that $T_2^n(\vec{c}) = 0$, we know $\mathrm{Res}(G,N)(\vec{c}) = 0$. In this case it is easy to prove by induction on length of paths that $\mathrm{Res}(K,N)(\vec{c}) = \mathrm{Res}(K,N')(\vec{c})$ for every $K \in \mathrm{Nodes}(N)$. Hence $\mathrm{Res}(G,N')(\vec{c}) \vee \mathrm{Res}(U,N')(\vec{c}) = 0$, and thus $\mathrm{Res}(U',N')(\vec{c}) = 0$. On the other hand, suppose $\mathrm{Res}(U',N')(\vec{c}) = 0$. Then $\mathrm{Res}(U,N')(\vec{c}) = \mathrm{Res}(G,N')(\vec{c}) = 0$ so $\mathrm{Res}(G,N)(\vec{c}) = 0$ since N and N' are identical on the predecessors of G. Thus, it is again easy to verify by induction that $\mathrm{Res}(K,N)(\vec{c}) = \mathrm{Res}(K,N')(\vec{c})$ for every node $K \in \mathrm{Nodes}(N)$, so $\mathrm{Res}(U,N)(\vec{c}) = 0$. Hence

$$(\mathrm{Res}(U,N)(\vec{c}) = 0) \leftrightarrow (\mathrm{Res}(U',N')(\vec{c}) = 0)$$

so $T_2^n = \mathrm{Res}(U,N) = \mathrm{Res}(U',N')$.

The effect of this rewiring is to remove G' from any path which has more than 1 ∧-gate on it. Since the transformation decreases the number of ∧-gates with ∧-gate successors[†], by recursively repeating this process for top-level ∧-gates with ∧-gates below them, the result is a single-level M-circuit for T_2^n with r ∧-gates and t ∨-gates. Call this modified circuit N_0. We remark in passing that every path from an input to the output in N_0 must have exactly one ∧-gate on it since $PI(T_2^n)$ contains no single variables.

We can now extract the radix-join network from N_0. Let U denote the output gate of N_0, let $\mathcal{G} = \{G_1, G_2, \ldots, G_r\}$ denote the set of (incomparable) ∧-gates in N_0, and let $\mathrm{Pred}(G_i) = \{I_i, J_i\}$ for all $i \in [1:r]$. See Figure 5.17. There is a gate in $\mathcal{G}$ on every path from INPUTS(N) to U. Since N_0 is minimal, there is a path from each node of $\mathcal{G}$ to U. Since there are only ∨-gates on any path from $\mathcal{G}$ to U, we know that $\mathrm{Res}(U) = \bigvee_{G \in \mathcal{G}} \mathrm{Res}(G)$. Moreover, since N_0 contained a minimal number t of ∨-gates , we know that

[†] Observe that for arbitrary nodes A and B of N , if A is a member of $\mathrm{Succ}^*(B,N'')$, then A was a member of $\mathrm{Succ}^*(B,N')$.

the portion of the circuit between the $\wedge$-gates G and the output U consists of a tree of $\vee$-gates. Thus there are $r-1$ $\vee$-gates $H = \{H_1,\dots,H_{r-1}\}$ (one of which is U) below G in the circuit.

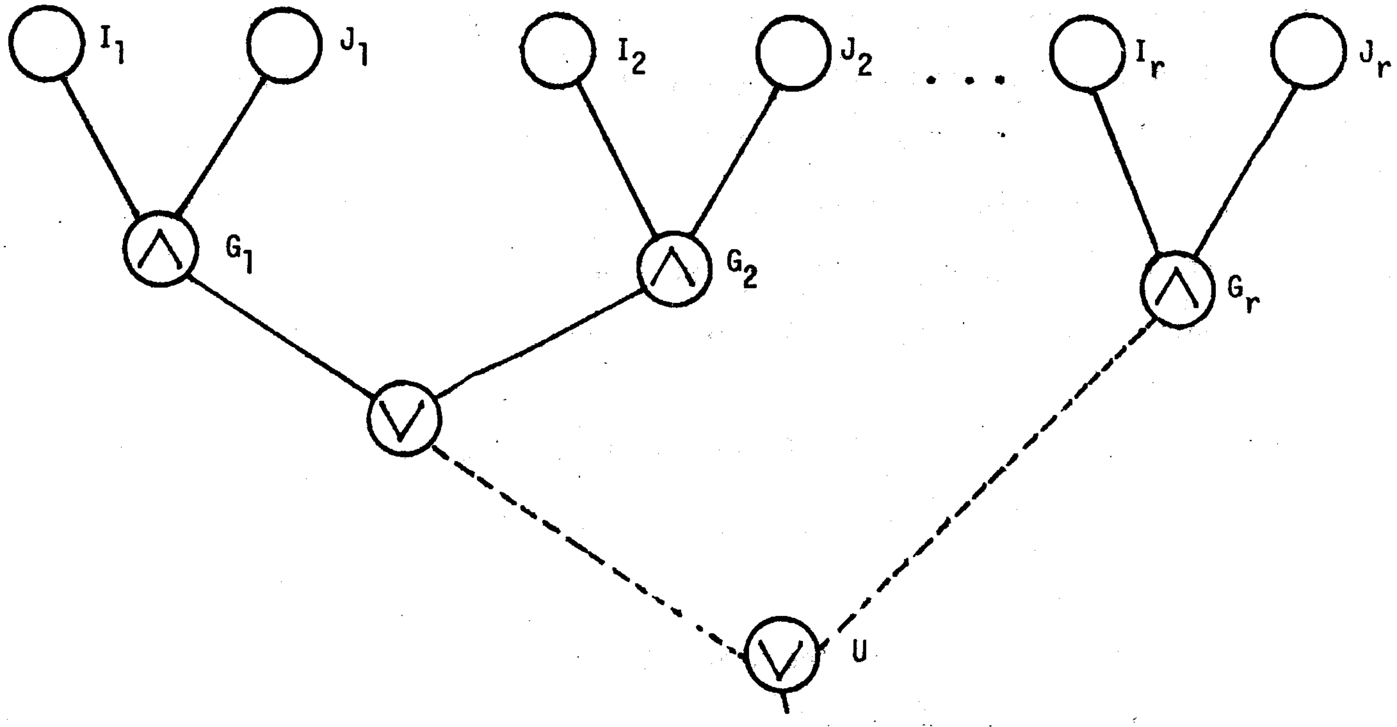


Fig. 5.17 The Circuit N_1

Fix $j \in [1:r]$ and let $i \in [0:n-1]$. By Corollary 5.10, since N_0 has $r = \log n$ $\wedge$-gates, we know that there is a partition of the input variables $X = \{x_k \mid k \varepsilon [0:n-1]\}$ into two sets A and B such that $\mathrm{Res}(I_j)$ is the disjunction of all variables in A and

$\mathrm{Res}(J_j)$ is the disjunction of all variables in B. Hence, since the circuit of the predecessors of G consists entirely of $\vee$-gates, there is a path from the particular variable x_i to exactly one of I_j and J_j.

Moreover, suppose that x_i and x_k are two distinct variables of X. Since $x_i x_k \in PI(T_2^n)$, and since $T_2^n = \bigvee_{G_j \in G} G_j$, there must be some gate G_j such that $x_i x_k \in PI(G_j)$. Hence there is either a path from x_i to I_j and a path from x_k to J_j or vice versa.

With these observations, we may renumber the inputs as follows: If E is an input node to N_0, we label E by x_i, where we define the index $i \in [0:n-1]$ for E bitwise by defining

$$(i)_j = \begin{cases} 1 & \text{if there is a path from } E \text{ to } I_j \text{ in } N_0 \\ 0 & \text{if there is a path from } E \text{ to } J_j \text{ in } N_0 \end{cases}$$

for each $j \in [1:r]$. By the above observations, it is easy to verify that, since $n = 2^r$, each input node gets labelled with a variable from $\{x_i \mid i \in [0:n-1]\}$. Moreover, each input node gets labelled with a distinct variable since two distinct input nodes must have paths to different predecessors of one of the $\wedge$-gates G_j. We assume w.l.o.g. that this is the numbering of the inputs since T_2^n is a symmetric function.

Now let R_n be the circuit obtained from N_0 by deleting the sets of gates G (all the $\wedge$-gates of N_0) and H ($r-1$ $\vee$-gates). Since R_n is an $\{\vee\}$-circuit, we know that, for any gate D in R_n,

$[x_i \epsilon PI(D)] \Leftrightarrow [D \epsilon Succ^*(x_i)]$. Hence, if the radix-join function F^n has components $(a_1, b_1, a_2, b_2 \ldots, a_r, b_r)$, then, for each $j \epsilon [1:r]$, $a_j = Res(J_j, R_n)$ and $b_j = Res(I_j, R_n)$. Hence R_n is a radix-join network for n-variables.

Since R_n contains $t - r + 1$ v-gates, the theorem is proved.

□ Theorem 5.16.

Armed with this theorem, we proceed to obtain bounds on the number of v-gates needed to compute the radix-sort function in {v}-circuits. We first prove a general lemma about {v}-circuits.

<u>Defintion</u>: Suppose N is an M-circuit. Define <u>the center of N</u>, denoted Z_N,by $Z_N = \{G \epsilon Gates(N) \mid Res(G,N) = \underline{0}\}$. Note that $Z_N = \emptyset$ unless outdeg(ZERO) ≥ 1.

<u>5.17 Lemma</u>: Suppose N is a {v}-circuit with inputs $\{x_1, x_2, \ldots x_n\}$ for which outdeg(ZERO,N) = 0 , and some subset A of inputs to N is set to 0, yielding a circuit N'. Then it is possible to remove from N' at least

$$\sum_{I \epsilon A} outdeg(I,N) + \sum_{G \epsilon Z_{N'}} (outdeg(G) - 1)$$

v-gates in $Succ^+(A,N)$.

Proof: Recall that the operation "setting a variable x to 0" in a circuit involves replacing x by ZERO in the predecessors of all gates in Succ(x). Hence $\text{outdeg}(\text{ZERO},N') = \sum_{I \in A} \text{outdeg}(I,N)$ and it suffices to show that at least

$$\text{outdeg}(\text{ZERO},N') + \sum_{G \in Z_{N'}} (\text{outdeg}(G) - 1) \qquad (4)$$

v-gates in $\text{Succ}^{+}(\text{ZERO},N')$ may be removed from N'. We prove this by induction on $|Z_{N'}|$.

If $Z_{N'} = \emptyset$, then every gate in $S = \text{Succ}(\text{ZERO},N')$ may be eliminated from N' since each gate in S has at least one input constant. If $|S| < \text{outdeg}(\text{ZERO},N')$, then by the pigeon-hole principle there must be some gate $G \in S$ with both nodes in $\text{Pred}(G,N')$ being the node ZERO. But then $G \in Z_{N'}$, which is a contradiction. Hence, at least $\text{outdeg}(\text{ZERO},N')$ gates may be eliminated.

Now assume that $|Z_{N'}| = k > 0$, and that (4) is true for all {v}-circuits with centers of size $k-1$. Let G_0 be a member of $I(Z_{N'})$, the set of initial gates of $Z_{N'}$, and suppose $\text{Pred}(G) = \{J,K\}$. See Figure 5.18. Since $\text{Res}(G_0,N') = \underline{0}$ and G_0 is an v-gate, we must have $\text{Res}(J,N') = \text{Res}(K,N') = \underline{0}$ by Lemma 2.1. Since J and K are not members of $Z_{N'}$ (by definition of G_0), it must be the case that $J = K = \text{ZERO}$. See Figure 5.19.

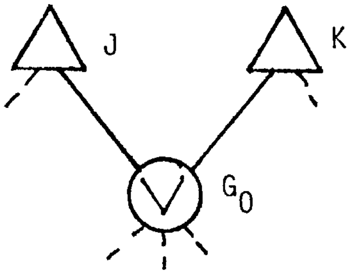


Fig. 5.18 $G_0 \in I(Z_{N'})$

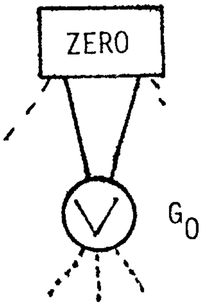


Fig. 5.19

We construct a new circuit $\bar{N}$ from N' by eliminating gate G_0, and replacing G_0 by ZERO in Pred(H) for every gate $H \in \mathrm{Succ}(G_0, N')$. For every node $L \in \mathrm{Nodes}(\bar{N})$, we know that $\mathrm{Res}(L,N') = \mathrm{Res}(L,\bar{N})$ since G_0 was replaced by a node which computes the same function. Furthermore, $|Z_{\bar{N}}| = |Z_{N'}| - 1$ (since G_0 was removed from the circuit); $\text{outdeg}(G,\bar{N}) = \text{outdeg}(G,N')$ for every gate $G \in Z_{\bar{N}}$; and $\text{outdeg}(\text{ZERO},\bar{N}) = \text{outdeg}(\text{ZERO},N') - 2 + \text{outdeg}(G_0,N')$. Thus, by induction, it is possible to remove from $\bar{N}$ at least an additional

$$\text{outdeg}(\text{ZERO},\bar{N}) + \sum_{G \in Z_{\bar{N}}} (\text{outdeg}(G,\bar{N}) - 1) =$$

$$\text{outdeg}(\text{ZERO},N') + [\sum_{G \in Z_{N'}} (\text{outdeg}(G,N') - 1)] - 1$$ gates from

$\text{Succ}^{+}(\text{ZERO},\bar{N}) \subseteq \text{Succ}^{+}(\text{ZERO},N')$, which, together with the elimination of G_0, proves the lemma.

□ Lemma 5.17.

One additional lemma will prove useful:

<u>5.18 Lemma</u>: Suppose $n \in \mathbb{N}$, $n \geq 4$, and $n \neq 2^r+1$ for any $r \in \mathbb{N}$. Let R_n be a minimal $\{v\}$-circuit for the radix-join function F^n. If $i \in [0:n-1]$, then $\text{outdeg}(x_i,R_n) \geq 2$.

<u>Proof</u>: Suppose n, R_n, and i are as in the statement of the lemma. Let $F^n = (a_1,b_1,\ldots a_{\lceil \log n \rceil}, b_{\lceil \log n \rceil})$. Observe that $|PI(a_1)| = \lceil n/2 \rceil$ and $|PI(b_1)| = \lfloor n/2 \rfloor$ and hence neither $PI(a_1)$ nor $PI(b_1)$ consists of a single variable. Since x_i is a member of one of $PI(a_1)$ or $PI(b_1)$, we must have that $\text{outdeg}(x_i,R_n) \geq 1$.

Suppose that $\mathrm{outdeg}(x_i,R_n) = 1$, and let $\mathrm{Succ}(x_i,R_n) = \{G\}$ where $\mathrm{Pred}(G,R_n) = \{x_i,H\}$. See Figure 5.20. Since N is a $\{\vee\}$-circuit

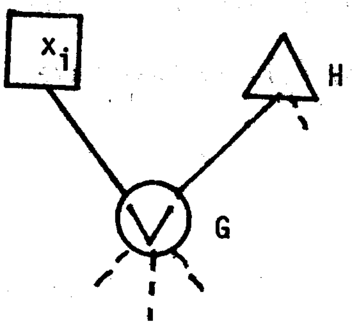


Fig. 5.20 The Case $\mathrm{outdeg}(x_i) = 1$

$PI(H,R_n)$ is a non-empty collection of single variables, so let $x_k \in PI(H,R_n)$ be chosen arbitrarily. Note that $k \neq i$ since the graph of R_n is acyclic.

It is easy to verify that in any $\{\vee\}$-circuit N,

$$[K \in \mathrm{Succ}^*(L,N)] \Rightarrow [PI(L,N) \subseteq PI(K,N)] \qquad (5)$$

for any $K,L \in \mathrm{Nodes}(N)$. Thus, in this case, since $\mathrm{Succ}^+(x_i,R_n) = \mathrm{Succ}^*(G,R_n)$, we must have $x_k \in PI(K,R_n)$ for any gate $K \in \mathrm{Succ}^+(x_i,R_n)$. However, if we let j be a radix position such that $(i)_j = (k)_j$ (which must exist since $i \neq k$), then one of

$PI(a_j)$ or $PI(b_j)$ should contain x_i but not x_k; assume w.l.o.g. that $x_i \epsilon PI(a_j)$ and $x_k \not\epsilon PI(a_j)$. Since R_n computes F^n, there must be a node D in R_n which computes a_j. Since $n \neq 2^r+1$, no component of F^n (including a_j) is the projection of a single variable, so D must be an $\vee$-gate. Since D depends on x_i and hence is in $Succ^+(x_i,R_n)$, we must have $x_k \epsilon PI(a_j)$ which is a contradiction. □ Lemma 5.18

We are now ready to establish a lower bound on the complexity of any $\{\vee\}$-circuit which computes the radix-join function.

<u>5.19 Theorem</u>: Suppose $n \epsilon \mathbb{N}$. Then $C_{\{\vee\}}(F^n) \geq 2n + 2\lfloor \log n \rfloor - 8$.

<u>Proof</u>: The theorem is easily verified for $n \leq 4$. For larger values of n, we first solve the problem in the case in which n is an exact power of 2, say $n = 2^r$ for some $r \epsilon \mathbb{N}$. We proceed by induction on r; the cases $r = 1$ and 2 have already been disposed of.

Assume the statement of the theorem true for $r - 1$, and that $n = 2^r$.

Suppose that $F^n = (a_1,b_1,\dots,a_r,b_r)$ and that R_n is a minimal $\{\vee\}$-circuit which computes F_n. Our general objective will be to show that by setting half of the variables $\{x_1,x_2,\dots,x_n\}$ to 0, one can eliminate at least $n + 2$ $\vee$-gates from the resulting circuit and have a circuit R' which computes $F^{n/2}$. Assuming this for the moment, we then know that

$$\begin{aligned} C_{\{v\}}(F^n) &= \hat{C}_{\{v\}}(R_n) \geq n + 2 + \hat{C}_{\{v\}}(R') \\ &\geq n + 2 + C_{\{v\}}(F^{n/2}) \\ &\geq n + 2 + 2(n/2) + 2(\log (n/2)) - 8 \\ &\qquad\qquad\qquad \text{(by induction)} \\ &= 2n + 2 \log n - 8, \end{aligned}$$

thus proving the theorem.

So suppose R_n is as above, and that I_k and J_k are the nodes in R_n which compute a_k and b_k respectively, for $k \in [1:r]$.

By fact (5) of page 145, in R_n each output node in $\{I_1, J_1, \ldots, I_r, J_r\}$ must have outdegree zero since the sets of prime implicants of each component of F^n are incomparable. In particular, if D_1 and D_2 are arbitrary output nodes, then $|PI(D_1) \cap PI(D_2)| \leq n/4$. Each output node is an $\vee$-gate since each component of F^n has $n/2 > 1$ prime implicants. These are the only nodes in R_n with outdegree zero since R_n is minimal.

Let

$$A = PI(I_r, R_n) = \{x_i \mid (i)_r = 0\}$$

and

$$B = PI(J_r,R_n) = \{x_i \mid (i)_r = 1\}$$

Observe that $A \cap B = \emptyset$ and $|A| = |B| = n/2$. Let R_A (resp. R_B) denote the circuit obtained by setting all variables in A (B) to 0. It is easy to see that R_B , with output nodes $\{I_1,J_1,\ldots,I_{r-1},J_{r-1}\}$, is a radix-join network on $n/2$ variables. Similarly R_A with the same set $\{I_1,J_1,\ldots,I_{r-1},J_{r-1}\}$ of output nodes is a radix-join network of the remaining variables in B if the r^{th} bit of the index of each variable in B is changed from 1 to 0.

We show that $n+2$ gates may be eliminated from one of these circuits R_A or R_B. In R_A, the only gate with outdegree zero which computes the constant function $\underline{0}$ is I_r. By Lemmas 5.17 and 5.18, this means that at least $2\cdot(n/2)-1 = n-1$ gates in $Succ^+(A,R)$ may be eliminated from R_A (where the only node in Z_{R_A} being accounted for is I_r). In addition, since $outdeg(J_r,R_A) = 0$, J_r also may be eliminated since it is no longer an output gate in R_A. Since $J_r \notin Succ^+(A,R_n)$, it was not eliminated previously, so at least n gates may be eliminated from R_A.

In a similar fashion one may count at least n gates eliminated from R_B.

To show that at least two additional gates may be eliminated from R_A or R_B we must look at other gates in the circuit. Let $Pred(I_r,R_n) = \{K_A,L_A\}$ and $Pred(J_r,R_n) = \{K_B,L_B\}$. See Figure 5.21.

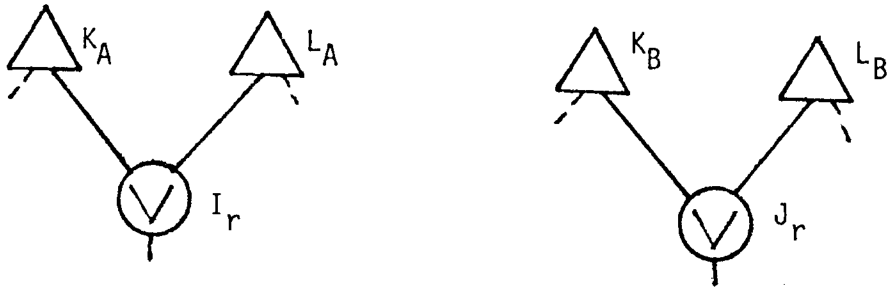


Fig. 5.21 The Predecessors

We consider several cases:

<u>Case 1(a)</u>: K_A and L_A are v-gates with outdegree in R_n more than one. See Figure 5.22.

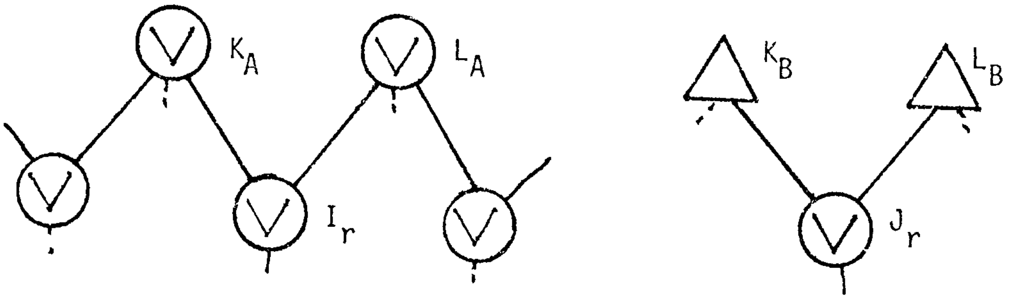


Fig. 5.22 Case 1(a)

In this case, two additional gates in R_A may be eliminated by Lemma 5.17. Both K_A and L_A are members of Z_{R_A} and both are of outdegree at least two in R_N. The count above of n gates considered only those members of Z_{R_A} of outdegree zero.

<u>Case 1(b)</u>: K_B and L_B are both v-gates of outdegree in R_n more than one.

If this is the case, then in R_B two additional gates may be eliminated; the proof is handled as in 1(a) with A replaced by B.

<u>Case 2(a)</u>: K_A and L_A are both v-gates of outdegree one in R_N. See Figure 5.23

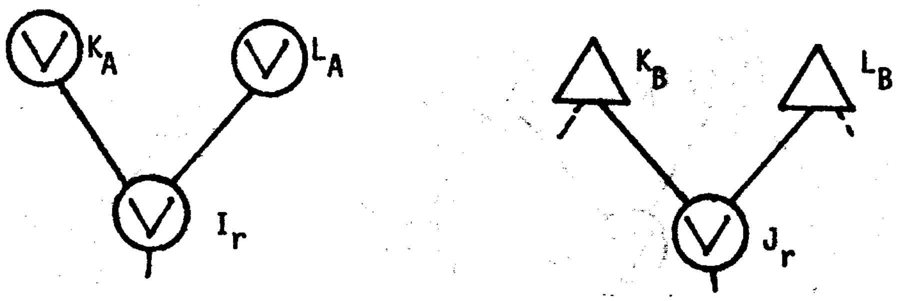


Fig. 5.23 Case 2(a)

In this case, both K_A and L_A in addition may be eliminated from R_B. We have previously eliminated from R_B the unique gate I_r in $\text{Succ}(K_A,R_n) = \text{Succ}(L_A,R_n)$. Since neither K_A nor L_A is an output gate in R_B, each may also be eliminated from R_B.

<u>Case 2(b)</u>: K_B and L_B are both v-gates of outdegree one in R_N.

Proved similarly to 2(a).

<u>Case 3(a)</u>: Cases 1 and 2 do not hold; one of $\{K_A,L_A\}$ is an v-gate of outdegree more than one in R_n; and one of $\{K_B,L_B\}$ is an v-gate of outdegree one in R_n. (The other nodes might possibly be inputs.) See Figure 5.24.

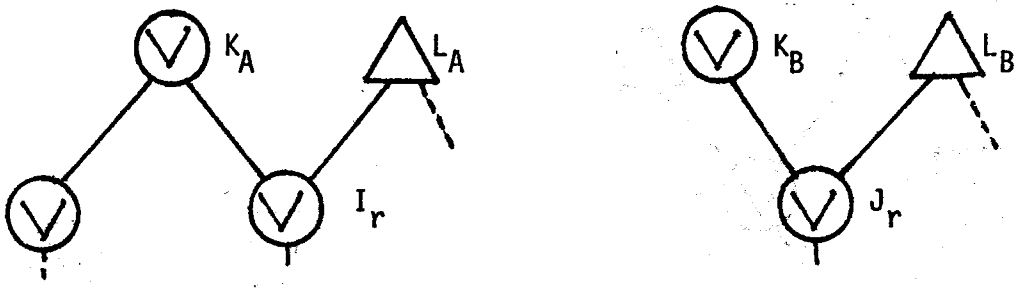


Fig. 5.24 Case 3(a) -- One Possibility

In this case two additional gates v-gates may be eliminated from R_A. Z_{R_A} has at least one gate of outdegree at least two since both K_A and L_A are members of Z_{R_A}, so Lemma 5.17 allows us to eliminate at least one additional gate. The v-gate of outdegree one in $\{K_B, L_B\}$ may also be eliminated since its unique successor J_r had been previously eliminated and since neither K_B and L_B is an output gate in R_A.

Case 3(b): Same as Case 3(a) with A and B interchanged. Proof is similar.

Case 4: One of $\{K_A, L_A\}$ is an input, the other is an v-gate of outdegree one; one of $\{K_B, L_B\}$ is an input, and the other is an v-gate of outdegree one.

Suppose w.l.o.g. that K_A and K_B are inputs. Let $\text{Pred}(L_A, R_n) = \{M, N\}$. See Figure 5.25. At least one of M or N must

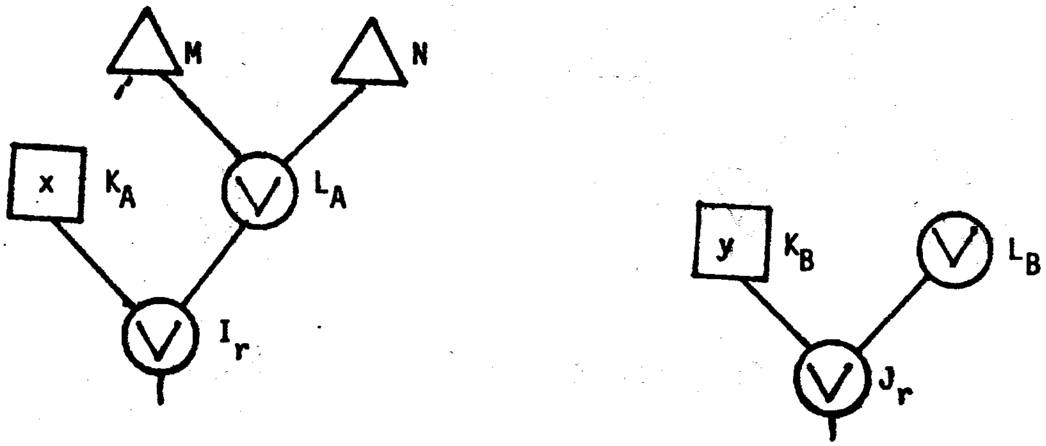


Fig. 5.25 Case 4

be an $\vee$-gate, for if both M and N were inputs, then $|PI(I_r;R_n)| = 3$; but $|PI(a_r)| = (n/2) \geq 4$ so this is impossible. Suppose w.l.o.g. that M is an $\vee$-gate.

If $\text{outdeg}(M,R_n) \geq 2$, then $M \in Z_{R_A}$, so in R_A one additional gate in $\text{Succ}^+(M,R_n)$ may be eliminated by Lemma 5.17. Also L_B may be eliminated since it is not an output gate and its unique successor has been eliminated. On the other hand, if $\text{outdeg}(M,R_n) = 1$, then in R_B both L_A and M may be eliminated in turn since each of their successors has been eliminated and neither is an output gate.

The above list of cases exhausts all possible arrangements of K_A, L_A, K_B, and L_B; the other arrangements are not possible in a minimal $\{\vee\}$-circuit. For example, it is impossible for K_A to be an input and L_A to be an $\vee$-gate of outdegree more than 1. If so then since $|PI(I_r,R_n)| = n/2$, $|PI(L_A,R_n)| \geq (n/2) - 1$. However, the set of prime implicants of any output other than I_r has at most $n/4 < (n/2) - 1$ variables in common with $PI(I_r,R_n)$, and hence the only output in $\text{Succ}^*(L_A,R_n)$ is I_r by fact (5) of page 145. Similarly, it is not possible for both K_A and L_A to be inputs.

This completes the proof of the theorem in the event n is an exact power of 2.

If n is not an exact power of 2, let $r = \lfloor \log n \rfloor$ and let $n_0 = 2^r$. Suppose that R_n is a minimal $\{v\}$-circuit for $F^n = (a_1, b_1, \ldots, a_{r+1}, b_{r+1})$ and let I_k and J_k be nodes in R_n computing a_k and b_k respectively (for $k \in [1:r+1]$). Let B be the set of variables $\{x_i \mid i \in [n_0:n-1]\}$ and observe that $PI(b_{r+1}) = B$. If R' is the circuit obtained by setting all variables in B to 0, then R', with outputs $\{I_1, J_1, \ldots, I_r, J_r\}$, is a radix-join network for n_0 variables.

If $n \neq n_0 + 1$, then by Lemma 5.18 each input $x_i \in B$ has outdegree at least two in R_n. Hence by Lemma 5.17 at least $2|B|-1$ gates in $Succ^+(B, R_n)$ may be eliminated from R' (since $J_{r+1} \in Z_{R'}$ is of outdegree zero). In addition, I_{r+1}, which is an v-gate since $|PI(a_{r+1})| > 1$, may also be eliminated from R' since it is an v-gate of outdegree zero which is not an output of R'. Since $|B| = n - n_0$, this means that a total of at least $2(n - n_0)$ gates may be eliminated from R'.

If $n = n_0 + 1$, then $PI(b_{r+1}) = \{x_{n-1}\}$. Since $x_{n-1} \in PI(a_1)$ and $|PI(a_1)| > 1$, we know that $outdeg(x_{n-1}, R_n) \geq 1$. Since N is minimal, J_{r+1} is the input node x_{n-1}. The setting of x_{n-1} to 0 results in a circuit R' from which at least one gate in $Succ^+(x_{n-1}, R_n)$ may be eliminated (since there are no gates of outdegree zero in $Z_{R'}$). Moreover, I_{r+1} is again a gate and may be eliminated from R' in addition to the gate previously mentioned, giving a total of two gates which may be eliminated from R'.

In either case, R′ may be simplified to be a radix-join network R′′ on n_0 variables with the elimination of $2(n-n_0)$ v-gates. Thus

$$
\begin{aligned}
C_{\{v\}}(F^n) = C_{\{v\}}(R_n) &\geq C_{\{v\}}(R'') + 2(n-n_0) \\
&\geq C_{\{v\}}(F^{n_0}) + 2(n-n_0) \\
&\geq 2n_0 + 2 \log n_0 - 8 + 2(n-n_0) \quad \text{(since } n_0 \text{ is a power of 2)} \\
&= 2n + 2r - 8
\end{aligned}
$$

and the theorem is proven.

□ Theorem 5.19.

5.20 Corollary: Suppose that $n = 2^r$, $r \in \mathbb{N}$, $r \geq 2$, and N is an M-circuit for T_2^n with exactly $\log n$ $\wedge$-gates. Then

$$\hat{C}_M(N) \geq 2n+4 \log n - 9.$$

Section D - Efficient Circuits for Threshold Functions

As mentioned, the proof of theorem 5.14 is a modification of a method first discovered by L. Adleman for computing threshold functions. A description of this method, which proves that $MC(T_k^n) \leq kn+o(n)$ for arbitrary fixed k, is to appear, and we sketch the technique briefly in this section for "threshold 2".

To compute T_2^n, the n variables are arranged in a $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ square matrix X. Let R_i be the disjunction of the variables of the i^{th} row of X, and let C_j be the disjunction of the variables of the j^{th} column. Then observe that

$$T_2^n(x_1,\ldots,x_n) = T_2^{\lceil\sqrt{n}\rceil}(R_1,\ldots,R_{\lceil\sqrt{n}\rceil}) \vee T_2^{\lceil\sqrt{n}\rceil}(C_1,\ldots,C_{\lceil\sqrt{n}\rceil})$$

since each pair of different variables in the matrix differ either in their row numbering or their column numbering. This construction give rise to the recurrence relations

$$MC(T_2^n) \leq 2n - 2\lceil\sqrt{n}\rceil + 1 + 2MC(T_2^{\lceil\sqrt{n}\rceil}) + 1$$

and $$MC(T_2^2) = 1$$

since the row and column sums can be constructed with at most $2n - 2\lceil\sqrt{n}\rceil + 1$ gates. This method shows that $MC(T_2^n) \leq 2n + 2\sqrt{n} + O(n^{1/4})$.

One problem with the above method is that it does not necessarily yield an M-circuit for T_2^n with the minimum number $\lceil\log n\rceil$ of $\wedge$-gates. For example, for $n = 8$ the resulting circuit has 4 $\wedge$-gates. The circuit for F^n described in Theorem 5.14 uses the same idea except that the variables are arranged into a rectangle with sides which are an exact power of 2. While the method of Theorem 5.14 yields an M-circuit with exactly $\lceil \log n \rceil$ $\wedge$-gates, it does so at the expence of increasing slightly the total number of gates. A close examination of the recurrence relationship for $C_{\{v\}}(F^n)$ shows that, in the notation of the proof of Theorem 5.14, either $n_1 \leq \sqrt{n}$ and $n_2 \leq \sqrt{2n}$, or $n_1 \leq \sqrt{2n}$ and $n_2 \leq \sqrt{n}$. Hence this method yields an M-circuit for T_2^n with at most $2n+(1+\sqrt{2})\sqrt{n} + O(n^{1/4})$ gates. (For some values of n the recurrence is better.) The table below describes the best known constructions for some small values of n.

n	Cor. 5.20 Lower Bound if log n $\wedge$-gates	Construction of Thms. 5.12 and 5.14		Adleman's Construction	
	(Total Gates)	(Total)	($\wedge$-Gates)	(Total)	($\wedge$-Gates)
2	-1	1	1	1	1
4	7	7	2	7	2
8	19	19	3	19	4
16	39	39	4	39	4
32	75	79	5	79	6
64	143	151	6	151	6
128	275	291	7	290	8

Table 5.1 The Complexity of T_2^n

Observe that for $n = 128$, the most efficient known M-circuit for T_2^n does not use the minimal number (log 128) of $\wedge$-gates.

Section E. Open Questions

There are a host of open questions raised by the above results:

(1) We know that $C_{\{v\}}(F^n) \le 2n + O(\sqrt{n})$. Determine $C_{\{v\}}(F^n)$ more exactly.

(2) Does an optimal M-circuit for T_2^n have exactly $\lceil \log n \rceil$ $\wedge$-gates?

(3) Is there a minimal M-circuit for T_2^n which is single-level? Is there one for an arbitrary quadratic function?

(4) Is there an M-circuit of size $c \cdot n \cdot \log n$ for T_k^n for arbitrary $k < n$ where c is independent of k? Is there an M-circuit of size $< \lceil \log k + 1 \rceil n$ for T_k^n for fixed k?

CHAPTER 6

The Complexity of Monotone Functions in Other Bases

The objective of this chapter is to study whether the complexity of a monotone Boolean function can be reduced by using a basis other than $\{\wedge,\vee\}$ in the formulas or circuits considered. That this is the case for Boolean circuits and multi-output functions is well-known and several examples exist. For instance, Paterson [1975] and Mehlhorn and Galil [1976], extending work originally done by Pratt[1974], have shown that the M-circuit complexity of the Boolean matrix multiplication of two $n\times n$ matrices is exactly $2n^3 - n^2$. On the other hand, Fischer and Meyer [1971] have shown that Strassen's fast integer matrix multiplication algorithm can be used to multiply two Boolean $n\times n$ matrices on a Turing Machine in time $O(n^{2.81\cdots})$. Using Fischer and Pippenger's results connecting time-bounded Turing Machine computations and circuits, this means that a circuit with $O(n^{2.82})$ gates for matrix multiplication exists in any complete basis[†]. Paul [1976] has improved this gap by exhibiting a series of monotone functions $f\in B_{n,n}$ for which $MC(f) = \Omega(n^2/(\log n)^{3/2})$ but for which $C_{B_2}(f) = O(n \log^2 n)$.

[†] Recall that the complexity over different complete bases is always related by a constant factor, hence it does not matter which complete basis is used.

It remains an open question as to whether such a gap between C_{B_2} and MC exists for a single-output function. Pippenger [1976] has shown that if one looks at the monotone function in B_n with the worst MC complexity among all m.b.f.'s in B_n, then its MC complexity is no more than a factor of $O(\log n)$ larger than its C_{B_2} complexity[+]. On the other hand, several researchers have conjectured that a large, even exponential gap may exist between the MC and M_{B_2} measures for a specific monotone Boolean function.

In contrast to this, it is possible to show that there are specific m.b.f.'s for which allowing a complete basis in the circuit does not allow a saving of more than a constant factor over the monotone case. This clearly the case for a function of minimal M-complexity which depends on all its arguments, and hence has M-circuit complexity of n-1. The situation can occur for more complex functions in addition. For example, in Chapter 3, we demonstrated that most quadratic m.b.f.'s of n variables have B_2 circuit complexity of $\Omega(n^2/\log n)$. Since each quadratic m.b.f. can be realized in an M-circuit which has $O(n^2/\log n)$

[+] In fact, Pippenger conjectures that among all monotone functions the worst case values for MC and for C_{B_2} are asymptotic, and offers a reward of $100 for proof of this conjecture.

gates, it is not possible to save more than a constant factor in circuit size for most quadratic functions.

We consider the formula size of functions in the remainder of this chapter. Here, the choice of complete basis will have a greater effect on complexity as pointed out by the work of Pratt [1975] - we consider the bases M, B_2, and U. One may make comments about the worst-case values for these measures for all monotone Boolean functions similar to those voiced for circuit complexities- it is the complexity relationship for individual functions which concerns us here. We show that this relationship can depend on the functions involved. Clearly, there are functions of complexity $n-1$ in all three measures for which no savings can be had. In work done jointly with M. Paterson, we do show that there are m.b.f.'s of M-formula complexity $\Theta(n_2/\log n)$ for which any formula in any complete basis must be of size $\Omega(n_2/\log n)$, and hence for which not more than a constant factor savings may be achieved. On the other hand, we exhibit a particular sequence of m.b.f.'s which have M-formula complexity $\Theta(n^2)$, but for which linear sized formulas exist over the basis B_2. This sequence is derived as a corollary of Khrapchenko's result that any U-formula for the parity function must have size n^2. While this example affirms that the complete basis B_2 may allow more succinct expressions for some m.b.f.'s, it does not answer the question of whether the addition of negation to the basis M allows a more compact expression for some functions. This latter question remains open.

Section A. A Case where a Complete Basis Doesn't Help

We will utilize a method introduced by Nečiporuk [1966] to establish a lower bound on the formula size of a particular monotone function. This method has been used by other authors in reference to other problems [Harper and Savage 1972, Paul 1977, and Lamagna 1975]. While the technique applies to formulas over arbitrary bases, we restrict our treatment here to the binary basis B_2.

Notation: Suppose $f \in B_n$ is an arbitrary Boolean function of the variables $x_1, \ldots, x_n$, and suppose $Y \subseteq \{x_i \mid i \in [1:n]\}$ is some subset of variables. Let $S(f,Y)$ denote the set of all restrictions of f to Y obtained by setting all remaining variables not in Y to be constant; that is,

$$S(f,Y) = \{f \mid_{x = c_x \text{ for } x \notin Y} \mid \vec{c}_x \in \{0,1\}^{n-|Y|} \}.$$

6.1 Lemma: Suppose f, Y, and S are as above. If F is any B_2-formula for f, and F contains $k \geq 2$ occurrences of Y variables, then

$$|S(f,Y)| \leq 16^{k-1}$$

Proof: We prove this fact by induction on k. Suppose f, Y, S, and k are as in the theorem.

In the event $k=2$, let x_i and x_j be the Y-variables occurring in F (where possibly $i = j$). Since no other Y-variables besides

x_i and x_j appear in F, all restrictions $f \mid_{\vec{x} = \vec{c}}$ for $x \notin Y$ of f to Y are functions of the variables $\{x_i, x_j\}$. Since there are at most 16 such functions, this proves the lemma in this case.

Now suppose that $k > 2$, and the lemma is true for all smaller values of k. In the tree corresponding to F, there exists a subtree of F with some number ℓ of occurrences of Y variables, where $2 \le \ell \le k-1$. This is true since the sons of a node in a binary tree create a partition of the leaves above the node. By passing upward[†] from the root of F, one may find such a subtree since each leaf contains at most 1 occurrence of a Y-variable. Thus we can decompose the formula F as

$$F(x_1,\ldots,x_n) = G(H(x_1,\ldots,x_n),x_1,\ldots,x_n) \qquad (1)$$

where H has ℓ occurrences of Y-variables and G has $k-\ell$ occurrences of Y-variables. Let $g(z,x_1,x_2,\ldots x_n)$ and $h(x_1,x_2,\ldots x_n)$ be Boolean fucntions defined by G and H respectively.

We may bound the number of restrictions of f to Y by bounding the number of restrictions of g and h. Let Y' denote the set of variables $Y \cup \{z\}$. Then G has exactly $k-\ell+1 < k$ occurrences of Y' variables. By (1), since any restriction of f to Y

[†] In a formula, as in all circuits, we think of the constant and input nodes as being at the top of the formula and arcs directed downward.

satisfies

$$f \mid_{\substack{\vec{x} = \vec{c} \\ \text{for } x \notin Y}} = g(h \mid_{\substack{\vec{x} = \vec{c} \\ \text{for } x \notin Y}}, x_1, x_2, \ldots, x_n) \Big|_{\substack{\vec{x} = \vec{c} \\ \text{for } x \notin Y}} ,$$

there can be no more restrictions of f to Y than the product

$$|S(h,Y)| \cdot |S(g,Y')| .$$

By induction, this implies that

$$|S(f,Y)| \le 16^{\ell-1} \cdot 16^{(k-\ell+1)-1} = 16^{k-1}$$

and the proof is complete. □ Lemma 6.1

<u>6.2 Theorem (Nečiporuk)</u>: Suppose $f \in B_n$ is a Boolean function and $P_1, P_2, \ldots, P_r$ is a partition of the variables $\{x_i \mid i \in [1:n]\}$ such that f depends on some variable in P_i for each $i \in [1:r]$. Then

$$L_{B_2}(f) \ge 1/4 \sum_{j=1}^{r} \log|S(f,P_j)| .$$

<u>Proof</u>: Suppose F is a minimal B_2-formula for f. For each $i \in [1:r]$, let k_i be the number of occurrences of P_i-variables in F. If $k_i \ge 2$, then $|S(f,P_i)| \le 16^{k_i-1}$ by Lemma 6.1, so $k_i - 1 \ge \log_{16}|S(f,P_i)| = 1/4 \log_2|S(f,P_i)|$. If $k_i = 1$, then $|S(f,P_i)| \le 4$, so $k_i \ge 1/4 \log |S(f,P_i)|$; the case $k_i = 0$ cannot

happen since f depends on some variable in P_i. In any event, $k_i \geq 1/4 \log |S(f,P_i)|$. Since $\hat{L}_{B_2}(F) = \sum_{i=1}^{r} k_i$, this proves the theorem. □ Theorem 6.2

We may now use this theorem to establish lower bounds on the complexity of particular m.b.f.'s. Note that if $n_i = |P_i|$ for $i \in [1:r]$, then $|S(f,P_i)| \leq 2^{2^{n_i}}$ since this is the number of Boolean functions of n_i variables, and $|S(f,P_i)| \leq 2^{n-n_i}$ since this is the number of settings of variables ~~outside~~ P_i. Choosing $n_i \sim \log n$ for each i gives a partition of the variables into $\sim n/\log n$ sets with a possible maximum value of $\log|S(f,P_i)|$ of $n - \log n$, to yield a theoretically possible $\Omega(n^2/\log n)$ lower bound using Theorem 6.2. It is possible to show that the maximal lower bound obtainable from the theorem is $\Omega(n^2/\log n)$.

Using an example similar to Nečiporuk's original function, we establish the following example:

<u>6.3 Theorem (Paterson, Bloniarz)</u>: For every n, there is an m.b.f. $f \in B_n$ such that

$$\text{(A)} \quad L_{B_2}(f) = \Omega(n^2/\log n)$$

and

$$\text{(B)} \quad ML(f) = O(n^2/\log n),$$

so that allowing a complete basis in a formula for f does not allow more than a constant factor reduction in size.

Proof: Suppose for the moment that $n = \binom{m}{(m+1)/2}$ where m is an odd integer. Note that m divides n. To define f, we regard the n input variables as arranged in a $(n/m)\times m$ rectangular matrix $\vec{x}_{ij}$. For each $i\in[1:n/m]$ and $j\in[1:m]$, we will fix

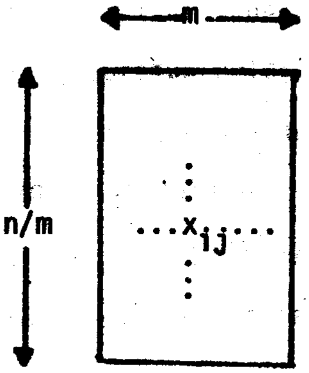


Fig. 6.1 The Inputs

σ_{ij} to be a distinct subset of $[1:m]$ of exactly $\binom{m}{(m+1)/2}$ elements (which is possible by our choice of n). Define $f:\{0,1\}^n \to \{0,1\}$ by

$$f(\vec{x}_{ij}) = \bigvee_{\substack{i\in[1:n/m]\\ j\in[1:m]}} [x_{ij} \wedge (\bigvee_{\substack{k\in[1:n/m]\\ k\neq i}} (\bigwedge_{q\in\sigma_{ij}} x_{kq}))] \qquad (2)$$

Intuitively, we may think of the σ_{ij}'s as row patterns -- $(m+1)/2$ entries which will be chosen from a row of m variables. If $k\in[1:n/m]$ and $\bigwedge_{q\in\sigma_{ij}} x_{kq}$ is true, we will say that row k satisfies pattern σ_{ij}. Note that this implies that row k has at least $(m+1)/2$ ones in it. The clause

$$\bigvee_{\substack{k\in[1:n/m] \\ k\neq i}} \left(\bigwedge_{q\in\sigma_{ij}} x_{kq}\right)$$

is one if some row other than the i^{th} satisfies σ_{ij}. Finally, $f(\vec{x}_{ij}) = 1$ if there is some variable x_{ij} equal to 1 whose corresponding pattern σ_{ij} is satisfied by some row other than the i^{th}. We note in passing that f is monotone by Theorem 2.4.

To obtain a lower bound using Theorem 6.2, consider the rows $P_i = \{x_{ij} \mid j\in[1:m]\}$ for $i\in[1:n/m]$ to be a partition of the input variables. We obtain a lower bound on $|S(f,P_i)|$. W.l.o.g. assume that $i = 1$; we show that a large number of settings of the remaining variables yield different restricted functions of f to P_1. In particular, suppose c_{ij} for $i\in[2:n/m]$, $j\in[1:m]$ are constants such that each row of constants has fewer than $(m+1)/2$ ones in it (that is, $|\{j \mid c_{ij} = 1\}| < (m+1)/2$ for each $i\in[2:n/m]$). We show that two different such settings of the variables $x_{ij} = c_{ij}$ for $i\in[2:n/m]$, $j\in[1:m]$ yield different restricted functions of the first row.

Let $f_{\vec{c}}$ denote the restriction $f\,|_{x_{ij}=c_{ij}}$ for $i\epsilon[2:n/m]$, $j\epsilon[1:m]$. The first row is the only possible row which can satisfy any pattern since each other row has fewer than $(m+1)/2$ ones in it. Hence the inner conjunction is false for $k \neq 1$, and

$$f_{\vec{c}} = \bigvee_{\substack{i\epsilon[2:n/m]\\ j\epsilon[1:m]}} (c_{ij} \wedge \bigwedge_{q\epsilon\sigma_{ij}} x_{1q})$$

Now if $\vec{c} \neq \vec{c}\,'$ are two distinct such constants, w.l.o.g. let i and j be indices such that $c_{ij} = 1$ and $c'_{ij} = 0$. Define an input for the first row by

$$x_{1q} = \begin{cases} 1 \text{ if } q\epsilon\sigma_{ij} \\ \\ 0 \text{ if } q\not\epsilon\sigma_{ij} \end{cases} \quad \text{for each } q\epsilon[1:m]$$

and note that the row x_{1q} satisfies pattern σ_{ij} and no other. Hence $f_{\vec{c}}(\vec{x}_{1q}) = 1$ whereas $f_{\vec{c}}'(\vec{x}_{iq}) = 0$, and the two functions $f_{\vec{c}}$ and $f_{\vec{c}'}$ are different on the input x_{1q}.

Thus $|S(f,P_1)|$ is at least the number of such settings of the remaining rows with fewer than $(m+1)/2$ ones per row. Since there are 2^{m-1} length m binary vectors with fewer than $(m+1)/2$ ones, we obtain

$$|S(f,P_1)| \geq [2^{m-1}]^{(n/m)-1} = 2^{n-(n/m)-m+1} \quad .$$

Hence, by theorem 2,

$$L_{B_2}(f) \geq 1/4 \sum_{i=1}^{(n/m)} [n-(n/m)-m+1]$$
$$= 1/4\ [(n^2/m) - (n^2/m^2) - n + (n/m)].$$

By Stirling's formula, [Knuth 1968, p. 46],

$$n = \binom{m}{(m+1)/2} \sim \sqrt{2/\Pi m} \cdot 2^m$$

so $m \sim \log n$. Hence $L_{B_2}(f) = \Omega(n^2/m) = \Omega(n^2/\log n)$ and the lower bound is established.

The upper bound on $ML(f)$ is obtained by expanding and factoring the definition (2) to get a more compact formula for f. Since f is not explicitly defined, we prove that an M-formula of size $O(n^2/\log n)$ exists without actually exhibiting it. Clearly, f has an M-formula of size $O(n^2)$, namely (2).

To get a smaller formula for f, we can distribute the variables in (2) to get the disjunctive form

$$f(x) = \bigvee_{\substack{i\in[1:n/m] \\ j\in[1:m]}} \ \bigvee_{\substack{k\in[1:n/m] \\ k \neq i}} (x_{ij} \wedge \bigwedge_{q\in\sigma_{ij}} x_{kq}) \ . \quad (3)$$

If we define, for $k \in [1:n/m]$,

$$f_k(\vec{x}) = \bigvee_{\substack{i \in [1:n/m] \\ j \in [1:m] \\ i \neq k}} \left(x_{ij} \wedge \bigwedge_{q \in \sigma_{ij}} x_{kq} \right), \qquad (4)$$

then by reversing the indexing in (3) we obtain

$$f(\vec{x}) = \bigvee_{k=1}^{n/m} f_k(\vec{x})$$

so $ML(f) \leq \sum_{k=1}^{n/m} ML(f_k)$. To bound $ML(f_k)$ for a fixed $k \in [1:n/m]$, note that each monome in $PI(f_k)$ consists of the distinct product of $(m+1)/2$ variables from row k (which has m variables in all) and a single variable not in this set. Since the number of prime implicants of f_k far exceeds m, by factoring (4) on the variables in row k we obtain a smaller formula for f_k.

<u>6.4 Lemma</u>: Suppose $r, s \in \mathbb{N}$, and $g \in B_{r+s}$ is a function such that

$$g(x_1, \ldots, x_r, y_1, \ldots, y_s) = \bigvee_{i=1}^{s} (y_i \wedge \bigwedge_{j \in A_i} x_j)$$

where, for each i, A_i is a unique subset of size a of $[1:r]$. Then

$$ML(g) \leq \binom{r+1}{a} + \binom{r}{a} - 1.$$

Proof of Lemma: We proceed by induction on r. If $r = 0$, then since there is a unique subset of size 0 of $\emptyset$, $g(y_1)$ is either y_1 or the constant function $\underline{0}$. In either case, $ML(g) \leq 1 = \binom{1}{0} + \binom{0}{0} - 1$.

If $r > 0$ and $a = r$, then clearly $g(x_1,\ldots,x_r,y_1) = y_1 \wedge x_1 \wedge \ldots \wedge x_r$ or $g = \underline{0}$, so $ML(g) \leq r+1 = \binom{r+1}{r} + \binom{r}{r} - 1$. Similarly, if $a = 0$, then again $ML(g) \leq 1 = \binom{r+1}{0} + \binom{r}{0} - 1$. Now suppose $0 < a < r$. By factoring the definition of g by x_r, we may express

$$g = \left(x_r \cdot g_1\right) \vee g_2 \qquad (5)$$

where

$$g_1 = \bigvee_{i \in T} (y_i \wedge \bigwedge_{j \in A_i - \{r\}} x_j)$$

and

$$g_2 = \bigvee_{i \in [1:s]-T} (y_i \wedge \bigwedge_{j \in A_i} x_j)$$

and T is the set of indices $i \in [1:s]$ such that $r \in A_i$. Note that each subset $A_i \setminus \{r\}$ for $i \in T$ has $a-1$ members chosen from $[1:r-1]$, and hence inductively

$$ML(g_1) \leq \binom{r}{a-1} + \binom{r-1}{a-1} - 1.$$

Similarly, each set A_i for $i \notin T$ has a members chosen from $[1;r-1]$, and hence

$$ML(g_2) \le \binom{r}{a} + \binom{r-1}{a} - 1.$$

Thus, by the expression (5),

$$\begin{aligned} ML(g) &\le 1 + ML(g_1) + ML(g_2) \\ &\le 1 + \binom{r}{a-1} + \binom{r-1}{a-1} - 1 + \binom{r}{a} + \binom{r-1}{a} - 1 \quad \text{(by induction)} \\ &= \binom{r+1}{a} + \binom{r}{a} - 1 \quad \text{(using the fact that } \binom{k}{\ell} = \binom{k-1}{\ell-1} + \binom{k-1}{\ell} \end{aligned}$$

[Knuth 1968, p. 54]).

□ Lemma 6.4.

Now, sinceLemma 6.4applies to $f_k(\vec{x})$ with $r = m$ and $a = (m+1)/2$, we obtain

$$\begin{aligned} ML(f_k) &\le \binom{m+1}{(m+1)/2} + \binom{m}{(m+1)/2)} - 1 \\ &= \binom{m}{(m+1)/2} + \binom{m}{(m-1)/2} + \binom{m}{(m+1)/2} - 1 \\ &= 3n - 1. \end{aligned}$$

Hence $ML(f) \le \sum_{k=1}^{n/m} (3n-1) = 3n^2/m - n/m = O(n^2/\log n)$.

To finish the proof, for n not of the form $\binom{m}{(m+1)/2}$ (m odd), we could choose n_0 of this form with $(n/4) \le n_0 \le n$ and merely ignore all but the first n_0 of the n arguments.

□ Theorem 6.3

Section B. A Case where B_2 Can Help.

In a counterpoint to the previous case, we present an example where the larger complete basis B_2 can allow a reduction in formula size of more than a constant factor. The example is a monotone cover of the parity function.

Definition: Suppose $f \epsilon B_n$ is an arbitrary Boolean function. An m.b.f. $g \epsilon B_{2n}$ is said to be a monotone cover of f if $f(x_1,\ldots x_n) = g(x_1, \neg x_1, x_2, \neg x_2,\ldots,x_n, \neg x_n)$.

Example: If $P^n(x_1,\ldots,x_n) = x_1 \oplus x_2 \ldots \oplus x_n$ is the parity function, then g^n defined by

$$g^n(x_1,y_1,x_2,y_2,\ldots,x_n,y_n) = [P^n(x_1,\ldots,x_n) \wedge \bigwedge_{i=1}^{n} (x_i \vee y_i)] \vee \bigvee_{i=1}^{n} (x_i \wedge y_i) \quad (6)$$

is easily verified to be a monotone cover of P^n.

We claim that the function g^n exhibits the properties desired. We use the following result of Khrapchenko [1971] which we state without proof.

6.5 Theorem: If $n \epsilon \mathbb{N}$, then

$$n^2 \leq L_U(P^n) < 2n^2 .$$

The fact that $L_U(P^n) < 2n^2$ is easily proved by recursively dividing $P^n(x_1,\ldots,x_n)$ as $(x_1 \oplus \cdots \oplus x_{\lfloor n/2 \rfloor}) \oplus (x_{\lfloor n/2 \rfloor +1} \oplus \cdots \oplus x_n)$. The lower bound is established by a remarkable argument which uses the isomorphism between series-parallel contact networks and U-formulas.

Assuming Theorem 6.5, it is straightforward to show that

$$n^2 \leq ML(g^n) \leq 2n^2 + 4n .$$

To derive the upper bound, suppose that F is a minimal U-formula for P^n. By applying DeMorgan's laws, we may distribute all negations in F to the variables without changing the size of the formula, so we assume w.l.o.g. that F has this property. Let F' be the formula obtained by replacing all occurrences of $\neg x_i$ in F by the literal y_i (for all $i \in [1:n]$). A straightforward argument demonstrates that

$$[F' \wedge \bigwedge_{i=1}^{n} (x_i \vee y_i)] \vee \bigvee_{i=1}^{n} (x_i \wedge y_i)$$

is an M-formula for $g^n(x_1,y_1,\ldots x_n,y_n)$.

For the lower bound on $ML(g^n)$, suppose that $F(x_1,y_1,\ldots,x_n,y_n)$ is any M-formula for g^n. By replacing y_i by $\neg x_i$ in F for each $i \in [1:n]$, we obtain a U-formula for P^n, which by Theorem 6.5 must have n^2 occurrences of literals. Thus F must have had size n^2.

Since P^n has a B_2 formula of size $n-1$, $L_{B_2}(g^n) \le 5n-1$. Since g^n has $2n$ variables, we have proven

<u>6.6 Theorem(Bloniarz, Meyer)</u>: For every $n \in \mathbb{N}$ there is an m.b.f. $f \in B_n$ such that

$$ML(f) = \theta(n^2)$$

and

$$L_{B_2}(f) \le 5n/2 + O(1) .$$

<u>Note</u>: Because of Theorem 6.5, any U-formula for g^n must also have at least n^2 occurrences of variables. Thus this example does not answer the question of whether the addition of negation to the basis B allows more compact expressions for m.b.f.'s.

<u>Section C. Open Questions</u>

(1) Is there any single-output m.b.f. for which $L_U(f) < ML(f)$; for which $C_{B_2}(f) < MC(f)$?

(2) Is there a larger gap than that given in Theorem 6.6 between ML and L_{B_2} ?

(3) Do larger gaps between these measures exist for multi-output functions?

CHAPTER 7

A Shortest-Path Algorithm

This final chapter is concerned with the problem of finding the shortest distance matrix for a non-negatively weighted directed graph. Several algorithms to solve this problem have been proposed [Dijkstra 1959, Floyd 1962, Spira 1973, Fredman 1976], the best worst-case running time being $O(n^3(\log\log n)^{1/3}/(\log n)^{1/3})$ [Fredman 1976]. However, Spira's algorithm, which has a worst-case time of $\Omega(n^3 \log n)$, has average running time $O(n^2 (\log n)^2)$.

Spira's algorithm basically searches for the nodes closest to a given source node. By searching along edges of minimum weight first, it is likely that the shortest paths from the source node to all other nodes will be discovered before all arcs in the graph need to be traversed. In fact, starting from any given source, on the average only $O(n \log n)$ arcs need to be traversed. By repeating this process for all n source nodes, the shortest path matrix can be found.

In Appendix 3 we show that several lacunae remain in Spira's algorithm. In particular, Spira does not specify any search pattern when there are paths from the source of equal length. We observe that arbitrary choices among paths of equal length do not solve the problem efficiently; inappropriate "tie-breaking" rules can result in algorithms which have $\Omega(n^3)$ running time on almost all matrices. One example of this phenomenom is presented in Appendix 3.

Our main result (this work was done jointly with A. Meyer and M. Fischer) is a correct version of Spira's algorithm, which we prove does indeed run in $O(n^2(\log n)^2)$ average time for a broad class of probability distributions on directed graphs with non-negative weights. These distributions properly include the distributions for which Spira originally made his claims. Informally the condition characterizing distributions for which our results hold is that information about the entire graph except for the arcs emanating from any given node i, together with information about the weights of the arcs emanating from node i, is not correlated with the assignment of the weights to those arcs -- all permutations of assignments are equally likely.

Finally, a modification of this algorithm is noted which computes the transitive closure of a Boolean matrix in average time $O(n^2 \log n)$. This result has recently been improved by Schnorr [1978a]who exhibits an $O(n^2)$ average-time algorithm for this problem.

Section A. Shortest Paths and Graph Distributions

Directed graphs were defined in Chapter 2. We henceforth assume that every n-node graph has a set of vertices $V = [1:n]$. A graph (V,E) is weighted if there is a cost function $c:E \to R \cup \{-\infty,\infty\}$ which assigns a weight or cost to each edge in the graph; the graph is non-negatively weighted if range $(c) \subseteq R^*$.

We consider only weighted graphs of this latter type and refer to them as weighted graphs.

One standard representation of a weighted graph is the $n \times n$ cost-matrix C, where $C(i,j)$ is defined to be the cost of edge (i,j) if edge $(i,j) \in E$, and $C(i,j) = \infty$ if $(i,j) \notin E$[†]. One additional representation will also be used. In the sorted list of edges representation, an n-node graph G is represented as a sequence of n adjacency lists. Each list element is a pair (a,w), where a is a vertex of G and w is a weight in R^*. The i^{th} list

$$L_i = ((a_{i1},w_{i1}),(a_{i2},w_{i2}),\ldots,(a_{ik_i},w_{ik_i}))$$

represents the k_i edges emanating in G from node i (including possibly an edge from i to itself); a_{ij} is the endpoint of the arc and w_{ij} is its corresponding weight. In addition, we require that the weights in each adjacency list be arranged in increasing order $w_{i1} \leq w_{i2} \leq \ldots \leq w_{ik_i}$[††]. In the case where

[†] We do not distinguish between missing edges and edges of weight ∞.

[††] The definition of $\leq$ and $+$ on R^+ are extended to R^* in the usual manner.

there are multiple edges in the adjacency list of the same weight, we assume that the endpoints are placed in the list in some prescribed order; for definiteness we assume that they are in order of increasing node index.[†]

We will represent the adjacency lists as two $n \times n$ matrices - the endpoint matrix A and the edgecost matrix W, where

$$A(i,k) = \begin{cases} a_{ik} & \text{if } k \le k_i \\ \text{NIL} & \text{if } k > k_i \end{cases}$$

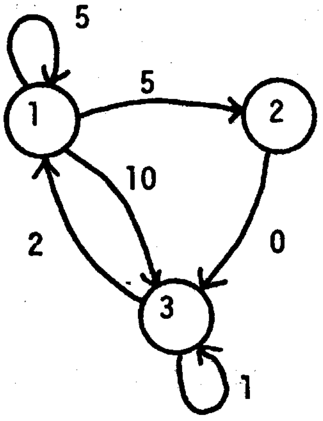
and

$$W(i,k) = \begin{cases} w_{ik} & \text{if } k \le k_i \\ \infty & \text{if } k > k_i \end{cases}$$

(NIL is a special symbol). Figure 7.1 exhibits different representations of a particular weighted graph.

Suppose $p = (v_1, v_2, \ldots, v_k)$ is a path in G, that is, a sequence of $k \ge 1$ nodes such that $(v_i, v_{i+1}) \in E$ for $i \in [1:k-1]$. The <u>cost of path p</u> is the sum of the costs of its edges; that is, $\sum_{i=1}^{k-1} c((v_i, v_{i+1}))$, where the cost of the trivial path (v) from v to v is 0. The <u>minimum cost matrix</u> M of G is the $n \times n$ matrix which has $M(i,j)$ equal to the minimum of the costs of all paths in G from i to j. The <u>shortest distance problem</u> is that of computing the minimum cost matrix M given the cost-matrix C of a weighted graph.

[†] This assumption is actually unnecessary for either the correctness or the timing of our algorithm.



Graph

$$\begin{bmatrix} 5 & 5 & 10 \\ \infty & \infty & 0 \\ 2 & \infty & 1 \end{bmatrix}$$

Cost-Matrix

$$A: \begin{bmatrix} 1 & 2 & 3 \\ 3 & \text{NIL} & \text{NIL} \\ 3 & 1 & \text{NIL} \end{bmatrix} \qquad W: \begin{bmatrix} 5 & 5 & 10 \\ 0 & \infty & \infty \\ 1 & 2 & \infty \end{bmatrix}$$

Sorted List of Edges

Fig. 7.1 Representations of a Graph

G_n will denote the set of all n-node weighted graphs. The subscript n will usually be omitted. Through the cost-matrix representation of graphs, we will identify G with $[R^*]^{n^2}$. A map $\Pi : G \to G$ is a <u>row permutation</u> if it simply permutes the endpoints of the edges emanating from a specified node while leaving the remainder of the graph intact; formally, there must be a permutation ρ of $[1:n]$ and an index $i_0 \in [1:n]$ such that, if G is any graph in G with cost-matrix C, and

C_{Π} is the cost matrix of $\Pi(G)$, then

$$C_{\Pi}(i,j) = \begin{cases} C(i,j) \text{ if } & i \neq i_0 \\ C(i,\rho(j)) \text{ if } i = i_0 \end{cases}$$

for every $(i,j)\in[1:n]^2$. The set of row permutations generates a group T_n of transformations on G; each member of T_n is called an adjacency transformation.

If P is a probability measure on G, its distribution function is the function $F_P:[R^*]^{n^2} \to \{x\in R \mid 0 \le x \le 1\}$ defined by

$$F_P(X) = P\{C\epsilon G | C(i,j) \le x_{ij} \quad \text{for} \quad (i,j)\epsilon[1:n]^2\}$$

where X is any element $(x_{ij} \mid (i,j)\epsilon[1:n]^2)$ of $[R^*]^{n^2}$. Every probability distribution is uniquely characterized by its distribution function.

We will say that a probability measure P on G is adjacency invariant (A.I.) if any adjacency transformation $\Pi\epsilon T_n$ is a measure preserving transformation on G; that is, $F_P(\Pi(x)) = F_P(x)$ for all $x\epsilon[R^*]^{n^2}$. We will consider specific examples of continuous A.I. probability measures later in this chapter. For a discrete probability measure on G, note that the measure is A.I. if and only if the probability of any graph is equal to the probability of the graph obtained by permuting the endpoints of the edges emanating from an arbitrary node in the graph.

For any function $f: G \to R$ and any probability measure P on G, let $E_P(f)$ denote the expected value of f with respect to P.

Section B. The Algorithm

Before describing the algorithm, we make an observation about weighted graphs which serves as basis for the algorithm. Suppose that G is a weighted directed graph, and that i is an arbitrary "source" node of G. Define, for any node $j \in \text{NODES}(G)$, $D(j)$ to be the minimum of the costs of all paths in G from i to j, where $D(j)$ is ∞ if there is no path from i to j. Let NEAR be a subset of the nodes of G such that $i \in \text{NEAR}$; those nodes in NEAR are called near nodes, and all other nodes are called far nodes. For each near node j, we define REALK(j) to be the far node k of G such that $C(j,k)$, the cost of the edge from j to k, is minimal among the costs of any edge from j to a far node. If j has several edges of equal minimal cost emanating from it, then REALK(j) is arbitrarily selected to be one of them. REALK(j) is undefined if j has no arcs to any far node.

We make the following observation.

7.1 Lemma: Suppose that G is a directed graph, and that NEAR satisfies the property that for every $j \in \text{NEAR}$ and $k \notin \text{NEAR}$, $D(j) \leq D(k)$. Let j_0 be a near node such that the sum

$D(j_0) + C(j_0, \mathrm{REALK}(j_0))$is minimal among all near nodes, and let $k_0 = \mathrm{REALK}(j_0)$. Then

(1) $D(k_0) = D(j_0) + C(j_0,k_0)$,

and (2) $D(k_0) \le D(k)$ for every $k \notin \mathrm{NEAR}$.

<u>Proof</u>: <u>Part(1)</u>. Since $D(j_0)$ is the cost of some path from i to j_0, we know that $D(j_0) + C(j_0,k_0)$ is the cost of a path from i to k_0, and hence $D(k_0) \le D(j_0) + C(j_0,k_0)$. Suppose to the contrary that $D(k_0) < D(j_0) + C(j_0,k_0)$, and let $i = v_1 ,v_2,\ldots,v_\ell = k_0$ be a path of weight $D(k_0)$ from i to k_0 Observe that $\ell \ge 1$ since $i \in \mathrm{NEAR}$. Let ℓ_0 be the least index in $[1:\ell]$ such that $v_{\ell_0} \notin \mathrm{NEAR}$; observe that $1 < \ell_0 \le \ell'$. Thus $v_{\ell_0-1} \in \mathrm{NEAR}$, and hence

$$D(v_{\ell_0-1}) + C(v_{\ell_0-1},\mathrm{REALK}(v_{\ell_0-1}) \le D(v_{\ell_0-1}) + C(v_{\ell_0-1}, v_{\ell_0}) \quad \text{(by def. of REALK)}$$

$$\le \sum_{m=1}^{\ell_0-1} C(v_m,v_{m+1}) \quad \text{(since } v_1,\ldots,v_{\ell_0-i} \text{ is a path to } v_{\ell_0-1})$$

$$\le \sum_{m=1}^{\ell-1} C(v_m,v_{m+1})$$

$$= D(k_0)$$

$$< D(j_0) + C(j_0,k_0).$$

But this latter inequality contradicts our choice of j_0 and part (1) is proven.

Part (2). Suppose to the contrary that k is a far node such that $D(k) < D(k_0)$. Let $i = w_1, w_2, \ldots, w_\ell = k$ be a path of length $D(k)$ from i to k. Again choosing ℓ_0 to be the least index in $[1:\ell]$ such that $w_{\ell_0} \notin$ NEAR, we must have $1 < \ell_0 \leq \ell$ and $w_{\ell_0 - 1} \in$ NEAR. Thus

$$\begin{aligned} D(w_{\ell_0-1}) + C(w_{\ell_0-1}, \text{REALK}(w_{\ell_0-1})) &\leq D(w_{\ell_0-1}) + C(w_{\ell_0-1}, w_{\ell_0}) \\ &\leq \sum_{m=1}^{\ell-1} C(w_m, w_{m+1}) \\ &= D(k) \\ &< D(k_0) \\ &= D(j_0) + C(j_0, k_0), \end{aligned}$$

which is again a contradiction to our choice of j_0.

□ Lemma 7.1

By the above argument, if NEAR satisfies the hypotheses of Lemma 7.1, then so does NEAR$\cup\{j_0\}$.

We may use Lemma 7.1 as basis for an algorithm for constructing the minimum cost matrix M for a graph G. We first present a version of the algorithm which uses certain simple operations on finite sets. We later note how these set operations can be implemented in terms of standard operations and data structures on random-access computers.

We assume that the graph is presented in its sorted list of edges representation as matrices A and W. Extraction of A and W from the cost matrix C may be performed by sorting the rows of C in a stable fashion[†] using edge weights as keys. The algorithm constructs the minimum cost matrix M one row at a time; this corresponds to finding the shortest distance from one particular node, say node i, to all other nodes in the graph. The algorithm searches from node i in a manner dictated by the weights of the edges; shorter paths are searched first.

In the algorithm, NEAR and D are as described in the hypotheses of Lemma 7.1 with one exception - D(j) is only defined for near nodes. If j is a far node, then D(j) = NIL. Initially NEAR is set to {i} and D(i) to 0. The algorithm halts with NEAR equal to the set of nodes which are reachable by paths from i.

[†] A sort is stable if the relative order of elements with keys of equal weightis preserved at the end of the sort (cf. Knuth [1973 Ch. 5]).

Inductively, to add a new node to NEAR, the algorithm attempts to find a node j_0 satisfying the hypothesis of Lemma 7.1. A priority queue is used to direct this search (see [Aho, Hopcroft, and Ullman 1974] for definition). In the course of execution of this algorithm, certain edges on a near node's adjacency list will be "examined"; initially no edges in the graph have been examined. Inductively we assume that all edges previously examined point to near nodes, and that all examined edges on an adjacency list will consist of those edges of least cost. A pointer P(j) is maintained for each node j in [1:n]; its value is the index in matrices A and W of the next (least-cost) unexamined edge on j's adjacency list. P(j) is initially set to 1 for all nodes.

Each near node j has an additional value associated with it, which we will call KEY(j). For these nodes, KEY(j) is the sum of D(j) and the weight of the next unexamined edge from node j; that is, KEY(j) = D(j) + W(j,P(j)). Since the examined edges on j's adjacency list all point to near nodes, we know that $\text{KEY}(j) \le D(j) + C(j, \text{REALK}(j))$.

To add a new vertex to NEAR, the algorithm selects a near node j such that KEY(j) is minimal among all near nodes. Suppose that k = A(j,P(j)) (the endpoint of the least-cost unexamined edge from node j) is a far node. Then $j = j_0$ satisfies the conditions of Lemma 7.1 since

$$D(j) + C(j,k) = KEY(j) \leq KEY(j')$$

$$\leq D(j') + C(j', REALK(j'))$$

for any other near node j′, and since C(j,k) is the least cost of any edge from j to a far node(all examined edges point to near nodes). Hence k can be added to NEAR and D(k) can be set to KEY(j) and still preserve the inductive hypotheses of Lemma 7.1.

On the other hand, if node k were already a member of NEAR, then its shortest distance is already known, so nothing is done. In either case, since the edge from j to k = A(j,P(j)) has been examined, the pointer P(j) is moved to the next edge from j.

This search is continued until either all nodes are found to be in NEAR or until there are no further edges to examine, in which case it follows from Lemma 7.1 that NEAR is the set of all vertices which are reachable by paths from i(since a missing edge has weight ∞).

One additional requirement is imposed in order that the general algorithm given above have fast running time. If the node j in NEAR from which we are searching has several edges of equal weight, then these edges are considered in consecutive executions of the search loop. That is, if W(j,P(j)+1) = W(j,P(j)), then we require that the algorithm select j as the near node from which to search in the next execution of the search loop[†].

[†] Together with the specifications on the tie-breaking rules of the priority queue below, this requirement avoids the difficulties in Spira's algorithm noted in Appendix 3.

To select nodes with appropriate KEY values, we use a priority queue. Three priority queue operations are necessary. CLEARQ sets the queue to the empty state. MINQ returns and removes from the queue a pair (KEY(j),j) such that j is a node in NEAR for which KEY(j) is a minimum among all nodes in NEAR. If the queue is empty, MINQ returns the pair (NIL,NIL). INSERTQ (d,j) inserts node j into the priority queue and sets KEY(j) to d. In case there are several nodes in the queue with minimal KEY values, MINQ selects and returns any one of them. (We assume only that the queue operates so that the value returned by a MINQ operation depends solely on the previous sequence of queue operations.)

The fully specified algorithm, R_i, which computes the shortest distance D(j) from node i to node j, is given below in a PASCAL-like language, where ∞ is represented by NIL:

Algorithm R_i

```
FOR  L := 1 to n  DO D(L) := NIL ;
FOR  L := 1 to n  DO P(L) := 1;
NEAR := {i} ; D(i) := 0; J := NIL; CLEARQ ;
IF  W(i,1) ≠ NIL  THEN <KEY,J> := <W(i,1),i>;
Comment  If there is an edge emanating from  i, then KEY is set
         to the minimum weight of all edges from  i.
WHILE  J ≠ NIL  DO
Comment  J is the near node to be searched from.  If REALK is
         defined for some near node, then J ≠ NIL.
BEGIN REPEAT  K := A(J,P(J));
      Comment  K is a candidate for REALK(J).
         IF  K∉NEAR  THEN
         Comment  K is REALK(J).
            BEGIN NEAR := NEAR ∪ {K};
               D(K) :=  KEY;
               IF |NEAR| = n  THEN HALT;
               IF  W(K,1) ≠ NIL  THEN INSERTQ (KEY+W(K,1),K)
            END;
        P(J) := P(J) + 1
      UNTIL    W(J,P(J)) ≠ W(J,P(J) - 1);
      Comment  Examine all equal-weight edges emanating from  J
               at the same time.
      IF  W(J,P(J)) ≠ NIL THEN INSERTQ ( D(J) + W(J,(P(J)),J);
    <KEY,J> := MINQ
END.
```

To compute the shortest-distance matrix M, algorithm R_i is run for each $i \in [1:n]$.

Shortest-distance algorithm

```
FOR  I := i to n  DO
      BEGIN
          $R_i$;
          FOR  J := 1 to n  DO M(I,J) := D(J)
      END .
```

Section C. Analysis of the Algorithm

If G is a weighted graph, we will denote by $N_i(G)$ the number of times the test "IF $K \neq NEAR$" is executed by R_i on graph G; that is, $N_i(G)$ is the number of edges of G examined[†] by algorithm R_i. Note that only a fixed number of operations are performed by R_i between successive executions of this test, so that the total number of operations performed by R_i on graph G is proportional to $N_i(G) + 1$. Thus we try to estimate $N_i(G)$.

7.2 Theorem: For any $i \epsilon [1:n]$ and any adjacency invariant probability measure P on $\mathcal{G}$;

$$E_P(N_i) \lesssim n \log_e n$$

Proof: Suppose that $i \epsilon [1:n]$ and that P is an A.I. probability measure on $\mathcal{G}$. If Π is any adjacency transformation, then Π is a measure preserving transformation on $\mathcal{G}$. Hence $E_P(N_i) = E_P(N_i \circ \Pi))$. Taking the sum of such expectations over all adjacency transformations Π, we have

$$\begin{aligned} E_P\Big(\sum_{\Pi \epsilon \mathcal{T}_n} (N_i \circ \Pi)\Big) &= \sum_{\Pi \epsilon \mathcal{T}_n} (E_P(N_i \circ \Pi)) \\ &= \sum_{\Pi \epsilon \mathcal{T}_n} E_P(N_i) \\ &= |\mathcal{T}_n| \cdot E(N_i) \end{aligned}$$

[†] Formally, an edge from J is "examined" when the pointer $P(J)$ is set to the next entry past the edge in the adjacency matrix.

thus, to prove that $E(N_i) \lesssim n \log_e n$, it suffices to show that

$$E_P(\sum_{\Pi\in\mathcal{T}_n} (N_i \circ \Pi)/|\mathcal{T}_n|) \lesssim n \log_e n.$$

We will prove the stronger statement that, for any graph $G\in\mathcal{G}$,

$$\left[\sum_{\Pi\in T_n} N_i(\Pi(G))\right]/|\mathcal{T}_n| \lesssim n \; \mathrm{loq}_e \, n. \qquad (1)$$

So suppose that $G\in\mathcal{G}$ is fixed, and let $\mathcal{G}_1$ be the set $\{\Pi(G) \mid \Pi\in\mathcal{T}_n\}$. An easy argument demonstrates that the left-hand side of inequality (1) is simply the average value of N_i with respect to the probability measure P_1 which assigns each graph in $\mathcal{G}_1$ equal probability, and all other graphs probability zero. Hence we must show that

$$E_{P_1}(N_i) \lesssim n \log_e n. \qquad (2)$$

Proving inequality (2) requires analyzing the average behavior of algorithm R_i over all inputs $G\in\mathcal{G}_1$. A convenient way to carry out this analysis is to consider a probabilistic algorithm R_i' which has no inputs. R_i' is the same as R_i except that where R_i would reference inputs describing a graph G, R_i' generates the inputs randomly. Note that every graph in $\mathcal{G}_1$ has the same edge-weight matrix W as G; it is only the endpoint matrices A which differ among the members of G_1. Moreover, each possible endpoint matrix is equally likely, modulo the requirement that edges of equal weight in a row be arranged in increasing node order.

Hence, the probabilistic algorithm R_i' which we construct selects at random and with equal probabilities one of these possible endpoint matrices and then executes R_i on the graph chosen. The selection of the endpoint matrix A is not done initially but rather dynamically as R_i references A (which is done in left-to-right fashion across the rows of A). At the first reference by R_i to entry A(J,K) for any J and K in [1:n], a value of the endpoint A(J,K) is chosen. If J has several edges of equal weight emanating from it, then at the first reference to any of these a selection of the endpoints for all these edges of equal weight is made and entered in the matrix A in the proper order. In all cases, by selecting these endpoints from all possible unused endpoints in an independent fashion with equal probabilities, one guarantees that every graph in G_1 will be chosen with equal probability.

R_i' is obtained by replacing every reference A(J,K) in R_i by the following probabilistic procedure A'(J,K). In this latter procedure, C is an internal n×n matrix which A' uses to store the endpoints which have already been selected; C(J,K) is initially set to <u>NIL</u> for every J and K in [1:n]. L is the number of endpoints to be selected, and SET is a set variable which collects those endpoints. *RANDOM* is a probabilistic procedure which returns an element of $[1:n]$ independently and with equal probability. *MIN* returns the minimal value of any set.

Procedure A´

```
A´(J,K):  IF  $C(J,K) \neq$ NIL   THEN RETURN ($C(J,K)$)
          ELSE
           BEGIN
            L := $|\{K' \in [1:n] \mid W(J,K) = W(J,K')\}|$; Comment The number of edges
            SET := $\emptyset$;                                  to be selected.
            REPEAT
                  B := RANDOM;
                  IF $B \notin \{C(J,K') \mid K' \in [1:K-1]\}$
                        THEN SET := $SET \cup \{B\}$
           UNTIL  $|SET| = L$;
           Comment  SET is now a random selection of  L  un-used
                    endpoints of edges from  J.
            For $J' = J$  to $(J + L - 1)$ DO
              BEGIN
                 R := MIN(SET);
                 SET := $SET - \{R\}$;
                 $C(J',K)$ := R

              END;
             Comment  These  L  edges are inserted into the graph
                      weight  $W(J,K)$  in the proper order.
           RETURN ($C(J,K)$)
            END;
```

We make the following obvious remark.

7.3 Lemma: Let E_0 be the average number of times the test "if $K \notin NEAR$" is executed over all possible runnings of algorithm R_i'. Then $E_0 = E_{P_1}(N_i)$.

To analyze the probabilistic algorithm R_i', note that once a node becomes a member of $NEAR \cup SET$, it will become a member of NEAR before the next call of procedure MINQ. This is true since endpoints of equal-weight edges from J are considered in successive passes of the loop of R_i'. Hence, if $NEAR \cup SET = [1:n]$ at any point in the algorithm, then at most n further executions of the test "IF $K \notin NEAR$" will occur before R_i' terminates.

Now let E be the expected number of times the procedure RANDOM is called in the execution of R_i' until $NEAR \cup SET = [1:n]$. At any point in a particular execution of R_i', the number of times the test "IF $K \notin NEAR$" has been executed is at most equal to the number of times nodes have been selected by RANDOM since the endpoint of every examined edge was at some point returned by RANDOM. Hence $E_{P_1}(N_i) = E_0 \leq E + n$.

But E is nothing but the expected number of times nodes from the set [1:n] must be chosen randomly with repetitions and placed in the set $NEAR \cup SET$ until $NEAR \cup SET = [1:n]$. This number is well known to be asymptotic with $n \log_e n$ [Feller 1968, p. 225] so $E_{P_1}(N_i) \lesssim n \log_e n$.

□ Theorem 7.2.

Section D. Implementation

There are several efficient ways of implementing a priority queue on a random access machine (R.A.M.) [Aho, Hopcroft, and Ullman 1974]. In particular, one can use a heap and ensure that every execution of MINQ and INSERTQ takes $O(\log n)$ steps, and every CLEARQ takes $O(n)$ steps. Standard implementations allow computing membership in, and cardinality of, the set NEAR with fixed cost. Under these implementation assumptions the average number of instructive steps requfred to execute R_i (for any $i\varepsilon[1:n]$) on a R.A.M. is $O(n \log^2 n)$ for any A.I. probability on G.

Since the shortest distance problem is that of computing the shortest distance matrix M from the cost matrix C, we must include the cost of extracting matrices A and W from C. This can be done by sorting at a cost of $O(n^2 \log n)$ basic steps, and we have the following theorem.

7.4 Theorem: Implemented as above, the shortest distance algorithm take an average of $O(n^2 \log^2 n)$ basic steps on a R.A.M. over any A.I. probability measure on weighted graphs.

For may applications, one wants not only to compute the minimum distance between all pairs of nodes i and j, but also to find a path in the graph which achieves that minimum cost. A simple addition to algorithm R_i enables one to retrieve such a path. A one dimensional array PATH is introduced and PATH(ℓ) is initialized to NIL for every $\ell\varepsilon[1:n]$. When a far node k is added to NEAR by virtue of its being at the endpoint of an arc from the near node j, then PATH(k) is set to j. This fully specified version of algorithm R_i is given below:

Algorithm P_i

```
FOR L := 1 to n DO PATH(L) := NIL;
FOR L := 1 to n DO D(L) := NIL;
FOR L := 1 to n DO P(L) := 1 ;
NEAR := {i}; D(i) := 0; J := NIL; CLEARQ;
PATH(i) = i;
IF W(i,1) ≠ NIL THEN <KEY,J> := <W(i,1),i>;
WHILE J ≠ NIL DO
BEGIN REPEAT K := A(J,P(J));
          IF K∉NEAR THEN
             BEGIN D(K) := KEY;
                   PATH(K) := J;
                   NEAR := NEAR∪{K};
                   IF |NEAR| = n THEN HALT;
                   IF W(K,1) ≠ NIL THEN INSERTQ( KEY + W(K,1),K)
             END;
          P(J) := P(J)+1
   UNTIL W(J,P(J)) ≠ W(J,P(J) - 1);
   IF W(J,P(J)) ≠ NIL THEN INSERTQ(D(J) + W(J,P(J)),J);
   <KEY,J> := MINQ
END.
```

The paths from all source nodes may be stored in the obvious way.

A straight-forward inductive proof shows the following: Suppose $k \epsilon \text{NEAR}$ at some stage in algorithm P_i. Then if we define the sequence of nodes:

$$w_1 = k$$

$$w_\ell = \text{PATH}(W_{\ell-1}) \quad \text{for } \ell \geq 1 \text{ in } \mathbb{N},$$

and if ℓ_0 is the least index in $\mathbb{N}$ such that $w_\ell = i$, then

$$i = w_{\ell_0}, w_{\ell_0 - 1}, \ldots, w_2, w_1 = k$$

is a path of cost $D(k)$ from i to k.

A final application allows us to modify the algorithm to compute the transitive closure of an (un-weighted) directed graph. If G is such a graph, the <u>transitive closure matrix</u> $I^*(G)$ is an $n \times n$ Boolean matrix defined so that

$$I^*(G)(i,j) = \begin{cases} 1 & \text{if there is a path from } i \text{ to } j \text{ in } G \\ 0 & \text{otherwise .} \end{cases}$$

The transitive closure problem is that of computing $I^*(G)$ from the $n \times n$ <u>incidence</u> matrix $I(G)$, where

$$I(G)(i,j) = \begin{cases} 1 & \text{if there is an arc from } i \text{ to } j \text{ in } G \\ 0 & \text{otherwise.} \end{cases}$$

We may identify an unweighted directed graph $G = (V,E)$ with the weighted directed graph G' obtained by assigning weights of zero to all edges in E and weight ∞ to all missing edges. We make the observation that the shortest distance matrix entry $M(i,j)$ is equal to 0 iff $I^*(i,j) = 1$.

Hence, any shortest path algorithm may be used to solve the transitive closure problem. For algorithms R_i and P_i, since all KEY's are equal to zero, one may implement the priority queue in such a way that every insertion and deletion from the queue can be performed with constant cost. As pointed out in a previous paper [Bloniarz, Fischer, and Meyer 1976] this results in a transitive closure algorithm with $O(n^2 \log n)$ average time.

Section E. A.I. Distributions

We conclude by presenting several examples of adjacency invariant probability measures.

1. For each $i\epsilon[1:n]$, suppose P_i is a probability measure on R^*. Let P be the probability measure on G obtained by selecting each entry of the weight matrix $C(i,j)$ independently according to distribution P_i. Then P, being the product measure generated by the measures P_i, is A.I. This class properly includes all distributions claimed in Spira [1973].

2. Suppose R is some probability measure on an arbitrary set B, and $f:G_n \to B$ is a function which is invariant under each $\Pi\epsilon T_n$; that is, $f \circ \Pi(G) = f(G)$ for all $\Pi\epsilon\tau_n$ and $G\epsilon\mathcal{G}$. Then the induced probability measure $R \circ f$ on G is A.I. by construction. Some specific examples include cases in which the weights of edges might be specified but all choices of endpoints are equally likely. For example, in the discrete case, one might specify a distribution on the sum of the weights of the edges leaving each node, and specify that each graph with the same sums be equally likely. Or one might specify a distribution on the maximum weight of any edge in the graph and specify that each graph with the same maximum be equally likely.

Section F. Open Questions

1. Schnorr [1978a] has recently exhibited an $O(n^2)$ average-time algorithm for transitive closure which improves on the ideas above. His algorithm does not construct the paths between all pairs of nodes; does a transitive closure algorithm exist which does construct the paths (as the algorithm P_i does) and which has $o(n^2 \log n)$ average time?

2. Is there a larger lower bound to the worst-case running time of any shortest-path algorithm than the $\Omega(n^2)$ obtained by adversary arguments? Can a lower bound on the average-case running time of such algorithms be established? For a related paper, see [Yao, Avis, and Rivest 1977, Graham, Yao, and Yao 1978, and Yao and Rivest 1978].

3. The fastest known algorithm for computing the transitive closure requires $O(n^{2.81..})$ basic steps in the worst case [Fischer and Meyer 1971]. Can this algorithm be improved upon?† Can Fredman's (worst-case $O(n^3)$) shortest-distance algorithm be improved upon?

(Fredman [1975] has observed that Spira's algorithm may be modified to compute the shortest-distance matrix M with a average total number of comparisons of only $O(n^2 \log n)$ but at a significant increase in total running time when all operations are considered.)

† Recently, Pan [1978] has improved on this result and has announced (1979) an even further improvement.

APPENDIX 1

In this appendix we sketch a proof pointed out to the author by V. Chvatal[†] of the lower bound on $H(n)$ stated in Theorem 3.14. We use notation as in Chapter 3.

<u>Theorem</u>: Suppose $0<\varepsilon<1/8$. Then, for arbitarily large n, there is an n-node undirected graph G_ε such that $D(G_\varepsilon) \geq n-n^{(1-\varepsilon)}+1$. Hence $H(n) \geq n-n^{(1-\varepsilon)}+1$.

<u>Proof Sketch</u>: Suppose $0<\varepsilon<1/8$. We will show the existence of an n-node undirected graph G_ε (for arbitrarily large n) for which $D(G_\varepsilon) \geq n-n^{1-\varepsilon}+1$. First we make some observations.

Suppose that G is an arbitrary undirected graph, and let $C = (h_1,\ldots,h_m)$ be an exact cover of G by complete bipartite graphs $G_1,\ldots,G_m$. Suppose G_i is a complete k_i-ℓ_i bipartite graph for $i\in[1:m]$. Observe that if both k_i and ℓ_i are at least 2 for some $i\in[1:m]$, then G must contain a 4-cycle; that is, there must be nodes a,b,c, and d in G such that edges $\{a,b\}$, $\{b,c\}$, $\{c,d\}$, $\{d,a\}$ are in G. Hence if G were a graph with no 4-cycles, we may without loss of generality assume that each graph G_i is a $(1,\ell_i)$ complete bipartite graph. Let v_i

[†] Private communication, 1977. Recent research by Bermond and Chung [Bermond 1978] have shown that

$$n-n^{19/24+\delta} < H(n) < n - \log_3 n + O(1)$$

for arbitrary $\delta>0$.

be the node in $h_i(G_i)$ corresponding to the singleton set in G_i.

It is clear that $\{v_i \mid i\epsilon[1:m]\}$ is a node cover of G, that is, a subset of the vertices of G such that every edge in G contains at least one member of the set. Hence, $D(G)$ equals the size of the smallest node cover of G, as long as G contains no 4-cycles.

If $0<\varepsilon<1/8$ is arbitrary, then Erdos [1959] has proved the existence of an arbitrarily large graph G_ε with n nodes which has no 4-cycles, and for which every set of at least $n^{1-\varepsilon}$ nodes spans at least one edge. Hence any node cover for G_ε can omit at most $n^{1-\varepsilon}-1$ nodes, so $D(G_\varepsilon) \geq n-n^{1-\varepsilon}+1$.

□ Theorem.

APPENDIX 2

In this appendix we explore several questions in the M-circuit complexity of combinations of functions. A general theorem, due to Galbiati and Fischer, is presented in which a combination of functions f and g is considered in which f and g depend on one variable in common†. This general proof includes as a special case M. Fischer's result that $MC(f \times g) = MC(f) + MC(g)$ [Paul 1976]. Finally, we remark that these results also apply to the measures $MC_{\wedge}$ and $MC_{\vee}$. Additional research on combinations of functions which are the disjunction of variables has been reported by Lamagna [1975], Neciporuk [1971], and Tarjan [1976].

1. Theorem (Galbiati and Fischer):

Suppose n,m,k, and $\ell \in \mathbb{N}$, and $f \in B_{n+1,k}$ and $g \in B_{m+1,\ell}$ are m.b.f.'s.

Define $f \Delta g \in B_{n+m+1,\ k+\ell}$ by

$$f\Delta g(x_1,\ldots,x_n,y_1,\ldots,y_m,z) = (f(x_1,\ldots,x_n,z),g(y_1,\ldots,y_m,z)).$$

Then $MC(f\Delta g) = MC(f) + MC(g)$.

Proof: Clearly $MC(f\Delta g) \leq MC(f) + MC(g)$; we prove the reverse

† This result has been presented in [Galbiah and Fischer, 1978] since the writing of this paper.

inequality.

Suppose N .is a minimal M-circuit which computes $f\Delta g$. We will say a gate $G\epsilon Gates(N)$ is a <u>mixed</u> gate if G depends structurally on some variable from $X = \{x_1,\ldots,x_n\}$ and some variable from $Y = \{y_1,\ldots,y_m\}$. We show that it is possible to re-structure N to eliminate all mixed gates and result in a circuit N' which has no more gates than N and which still computes $f\Delta g$. If this is the case, then N consists of two disjoint circuits; one consists of $Succ^*(X,N)\cup\{z\}$ and contains a gate computing $f(\vec{x},z)$, and the other consists of $Succ^*(Y,N\ )\cup\{z\}$ and contains a gate which computes $g(\vec{y},z)$. Hence the total number of gates in N' (and therefore N) is at least $MC(f) + MC(g)$.

So let $MIX = \{G\epsilon Gates(N)|G$ is a mixed gate$\}$ and suppose $MIX \neq \emptyset$. Then $I(MIX) \neq \emptyset$, so we can select an arbitrary gate $G\epsilon I(MIX)$ and let $Pred(G) = \{H,J\}$. Since N is minimal, neither H nor J is a constant node. We will replace G with another gate which is not mixed and yet still have a circuit which computes $f\Delta g$.

Since H and J are not in MIX, but G is, either H depends solely on $X\cup\{z\}$ and J depends on $Y\cup\{z\}$, or vice versa; suppose w.l.o.g. that H depends on $X\cup\{z\}$ and J depends on $Y\cup\{z\}$.

We first consider the case that G is an $\wedge$-gate. In this case

$$PI(G,N) \subseteq PI(H,N)\cdot PI(J,N) \ ; \qquad (1)$$

that is, every prime implicant of G is a product of a prime implicant of H and a prime implicant of J. By Lemma 2.8 , since no component of $f \Delta g$ has a prime implicant which contains both a variable from X and one from Y, we may eliminate from PI(G,N) all such monomes which contain both a variable from X and a variable from Y, and still have a circuit which computes $f \Delta g$. We consider seveal cases.

<u>Case 1</u>: Neither PI(H,N) nor PI(J,N) contains the monome z.

In this case, every monome in PI(H,N) contains an X-variable and every monome in PI(J,N) contains a Y-variable. Hence, by (1), every monome in PI(G,N) contains both an X-variable and a Y-variable so G may be replaced by a node which computes the function $\bigvee \emptyset$, the constant function $\underline{0}$. Since this new network has one fewer gate (namely G) than N, this contradicts N's minimality and hence Case 1 can not occur.

<u>Case 2</u>: z is a member of both PI(H,N) and PI(J,N).

In this case, $z \in PI(G,N)$. By an argument similar to Case 1, one can also show that all monomes in PI(G,N) different from z contain both an X-variable and a Y-variable and hence may be eliminated from PI(G,N) while still leaving a circuit which computes $f \Delta g$. Hence G may be eliminated and replaced by the input node for the variable z, a contradiction to the minimality of N.

Case 3: z is a member of exactly one of $PI(H,N)$ or $PI(J,N)$.

Assume w.l.o.g. that $z \in PI(H,N)$. In this case, every monome of $PI(H,N)$ other than z contains an X-variable, and every monome of $PI(J,N)$ contains a Y-variable. Thus every monome of $PI(G)$ contains a Y-variable, and those which do not contain an X-variable in addition are of the form $z \cdot t'$, where $t' \in PI(J)$ (note that t may also contain the literal z).

So let N' be the circuit obtained by replacing H in $Pred(G,N)$ by the input node for variable z. Then $PI(G,N')$ consists of those monomes of $PI(G,N)$ which do not contain both an X-variable and a Y-variable, and we have a circuit N' which still computes $f \Delta g$ by Lemma 2.8. Since G is not mixed in N', we have reduced the number of mixed gates in N by one.

In the event that G is an $\vee$-gate, we use an argument dual to the one above to also replace G by a non-mixed gate. Recursively repeating the above construction for each initial mixed gate, we obtain a modified M-circuit for $f \Delta g$ with no mixed gates, and no more total number of gates than the original. Hence the theorem is proven. □ Theorem 1.

2. Corollary (Fischer): Suppose that $f \in B_{n,k}$ and $g \in B_{m,\ell}$ are m.b.f's. Then $MC(f \times g) = MC(f) + MC(g)$, and any optimal circuit for $f \times g$ contains no mixed gates.

Proof: Define $f'(x_1,\dots,x_n,z) = f(x_1,\dots,x_n)$ and $g'(y_1,\dots,y_m,z) = g(y_1,\dots,y_m)$. Then $MC(f') = MC(f)$ and $MC(g') = MC(g)$. Also $f'\Delta g' = f\times g$, so the first part of the corollary holds. Moreover since $f'\Delta g'$ does not depend on variable z, cases 2 and 3 of the proof of Theorem 1 can never hold for a minimal circuit, and hence no mixed gates can occur.

□ Corollary 2.

In a similar fashion to Theorem 1, one also proves that the following holds.

3. Theorem: Suppose that $f\in B_{n,k}$ and $g\in B_{m,\ell}$ are m.b.f.'s. Then

$$(1)\quad MC_\wedge(f\times g) = MC_\wedge(f) + MC_\wedge(g)$$

and

$$(2)\quad MC_\vee(f\times g) = MC_\vee(f) + MC_\vee(g).$$

The proof of part (1) is similar to that of Theorem 1; one considers an $\wedge$-minimal circuit for $f\times g$ which among all such also has a minimal number of $\vee$-gates. Part (2) holds by duality.

A similar result holds to Theorem 3 holds for $f\Delta g$.

Note: We observe that the analogous question to Theorems 1 and 2 when f and g have 2 variables in common is false as the example

$$f(z_1,z_2) = g(z_1,z_2) = z_1 \wedge z_2$$

demonstrates .

APPENDIX 3

In this appendix we present a version of Spira's algorithm for computing shortest distances in a graph [1973] which has $\Omega(n^3)$ average running time over a certain class of graphs. This counter-example, which was previously reported in [Bloniarz, Fischer, and Meyer 1976], reveals why we needed to revise, correct, and verify Spira's original approach.

Spira's algorithm operates by computing the distance from a particular "source" node i to all other nodes in the graph. By repeating the algorithm for each source node i, the shortest-distance matrix may be found. Two differences between this algorithm, which we call S_i, and the algorithm R_i of Chapter 7 are noted. The first is that, in S_i, no assumptions are made about the value returned by MINQ in the case in which there are nodes in the queue with equal KEY's; the queue might utilize information about the graph in breaking ties. This is a minor difficulty. The other difference is that when a near node has several edges of equal weight emanating from it, then these edges are not necessarily examined in successive passes of the algorithm. These incomplete specifications lead to the problems we describe later. A Pascal-like implementation of S_i is given below:

ALGORITHM S_i (Spira's Algorithm)

```
FOR L := 1 to n DO D(L) := NIL;
FOR L := 1 to n DO P(L) := 1;
NEAR := {i}; D(i) := 0; J := NIL; CLEARQ;
IF W(i,1) $\neq$ NIL THEN <KEY,J> := <W(i,1),i>;
WHILE J $\neq$ NIL DO
BEGIN K := A(J,P(J));
      IF K$\notin$NEAR THEN
         BEGIN NEAR := NEAR$\cup${K};
               D(K) := KEY;
               IF |NEAR| = n THEN HALT;
               IF W(K,1) $\neq$ NIL THEN INSERTQ(KEY + W(K,1), K)
         END ;
      P(J) := P(J) + 1;
      IF W(J,P(J)) $\neq$ NIL THEN INSERTQ( KEY + W(K,1),K);
     <KEY,J> := MINQ
END .
```

The proof that S_i correctly computes the shortest distance $D(j)$ from node i to node j is identical with that given in Chapter 7 for algorithm R_i.

To show that algorithm S_i can be implemented poorly, for simplicity we will restrict ourselves to the case in which all edge weights are either 0 or ∞. This corresponds to computing the transitive closure of a graph as mentioned in Chapter 7. In this case, every node in the priority queue has a KEY of zero, and the value returned by MINQ is an arbitrary pair in the queue.

We present an example in which the priority queue is implemented so that the value returned by MINQ does not utilize any information about the graph (i.e. the matrices A and W) other than that given it by previous INSERTQ operations. In particular, we maintain a first-in, first-out queue to store the nodes in the priority queue. A node is INSERT'ed at the end of the queue, and MINQ removes and returns the node at the beginning of the queue.

Theorem: Let P be the uniform probability distribution on n-node weighted directed graphs in which all edge-weights are zero or ∞. For any graph $G \epsilon G_n$, assume (as usual) that edges of equal weight in all adjacency lists are sorted by increasing node index. If $N_i(G)$ is the number of times the subroutine MINQ is called in executing S_i on G, then, when implemented as described above,

$$E_P(N_i) = \Omega(n^2)$$

Proof: A straightforward argument shows that an average graph under this probability distribution

(1) has at least n/3 edges emanating from each node

and (2) has a path (of weight 0) from every node to every other node in the graph.

It suffices to show that there is a constant $c > 0$ such that if G is a graph satisfying (1) and (2), then $N_i(G) \geq cn^2$.

So suppose G satisfies (1) and (2). Then, in the execution of S_i on G, node n will be placed in NEAR at some point. Hence, at some point in the execution we must have $K = n$; let j_0 be the value of J at the point (the edge from j_0 to n is being "examined" at that point). Since j_0 had at least $n/3$ edges from it, and since the edge to n is the last edge listed in the j_0th row of A and W, we know that j_0 must have been placed in NEAR, placed on the queue, and subsequently returned by MINQ at least $n/3$ times.

Following the ℓth time that j_0 is returned by MINQ, an edge emanating from j_0 is examined, and its endpoint $A(j_0,\ell)$ is added to the queue (unless that element had previously been placed on the queue). The queue discipline implies that $A(j_0,\ell)$ will be returned by MINQ inbetween successive returns of j_0 by MINQ, starting from the ℓth time that j_0 is returned by MINQ and continuing at least until either node n is placed in NEAR (after at least $n/3 - \ell$ further returns of j_0 by MINQ) or until every edge from $A(j_0,\ell)$ has been examined. Thus $A(j_0,\ell)$ is returned by MINQ at least $\min(n/3 - \ell - 1, n/3) = n/3 - \ell - 1$ times. Since this is true at least for $1 \leq \ell < n/3$, the total number of calls to MINQ is at least

$$\sum_{\ell=1}^{n/3-1} (n/3 - \ell - 1) = \Omega(n^2)$$

and the statement is proved.

□ Theorem.

Since this theorem is true for every $i \in [1:n]$, a transitive closure (or shortest path) algorithm implemented as above will have $\Omega(n^3)$ average time, contrary to Spira's original assertion.

BIBLIOGRAPHY

Aho, A.V., J.E. Hopcroft, and J.D. Ullman 1974. The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, Massachusetts.

Arlazarov, V.L., E.A. Dinic, M.A. Kronrod, and I.A. Faradzev 1970. "On Economical Construction of the Transitive Closure of a Directed Graph," Dokl. Akad. Nauk SSSR, Vol. 194, pp. 487-488 (English translation in Soviet Math. Dokl., Vol. 11, pp. 1209-10).

Bermond, J.C. 1978. Couverture des Aretes d'un Graphe par des Graphes Bipartis Complets. Research Report 10, Laboratoire de Recherche en Informatique, Université de Paris-Sud, Centred'Orsay, France.

Bloniarz, P.A., M.J. Fischer, and A.R. Meyer 1976. "A Note on the Average Time to Computer Transitive Closures," Proceedings of the Third International Colloquium on Automata, Languages and Programming, Edinburgh University Press, Edinburgh, Scotland, pp. 425-434.

Borodin, A. 1977. "On Relating Time and Space to Size and Depth," SIAM Journal on Computing, Vol. 6, pp. 733-744.

Cook, S.A. 1971. "The Complexity of Theorem Proving Procedures," Proceedings of the Third Annual ACM Symposium on Theory of Computing, pp. 151-158.

Dijkstra, E.W. 1959. "A Note on Two Problems in Connexion with Graphs," Numerische Mathematik, Vol. 1, pp. 269-271.

Erdös, P. 1959. "Graph Theory and Probability," Canadian Journal of Mathematics, Vol. 11, pp. 34-38.

Ehrenfeucht, A., and P. Zeiger 1976. "Complexity Measure for Regular Expressions," Journal of Computer and Systems Sciences, Vol. 12, pp. 134-146.

Fischer, M.J. 1974. Lectures on Network Complexity, University of Frankfurt, Germany.

Fischer, M.J. 1975. The Complexity of Negation-Limited Networks-A Brief Survey. Project MAC Technical Memorandum 65, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Fischer, M.J., and A.R. Meyer 1971. "Boolean Matrix Multiplication and Transitive Closure," Proceedings of the Twelfth IEEE Symposium on Switching and Automata Theory, pp. 129-131.

Fischer, M.J., A.R. Meyer, and M.S. Paterson 1975. "Lower Bounds on the Size of Boolean Formulas," Proceedings of the Seventh Annual ACM Symposium on Theory of Computing, pp. 37-44

Feller, W. 1968. An Introduction to Probability Theory and Its Applications, Vol. 1, Third Edition, John Wiley and Sons, New York.

Floyd, R.W. 1962 "Algorithm 97: Shortest Path," Comm. Assoc. Computing Mach., Vol. 5, p.345.

Fredman, M.L. 1976. "New Bounds on the Complexity of the Shortest Path Problem," SIAM Journal on Computing, Vol. 5, pp.83-89.

Fredman, M.L. 1975. "On the Decision Tree Complexity of the Shortest Path Problems," Proceedings of the Sixteenth IEEE Symposium on Foundations of Computer Science, pp. 98-99.

Galbiati, Giulia, and M.J. Fischer 1978. On the Complexity of 2-Output Boolean Networks, Technical Report 78-08-01, Department of Computer Science, University of Washington, Seattle, Washington.

Graham, R.L., A.C. Yao, and F.F. Yao 1978. Information Bounds are Weak in the Shortest Distance Problem, unpublished manuscript.

Hensel, G. 1964. "Nombre Minimal De Contacts de Fermeture Necessaires pour Realiser une Function Booleene Symmetrique de n Variables," C.R. Acad. Science Paris, Vol. 258, pp. 6037-6040.

Harary, F. 1969. Graph Theory. Adison-Wesley, Reading, Massachesetts.

Harper, L.H., W.N. Hsieh, and J.E. Savage 1975. "A Class of Boolean Functions with Linear Combinational Complexity," Theoretical Computer Science, Vol. 1, pp. 161-183.

Harper, L.H. and J.E. Savage 1972. "On the Complexity of the Marriage Problem," Advances in Mathematics, Vol. 9, pp. 299-312

Harrison, M.A. 1965. Introduction to Switching and Automata Theory. McGraw-Hill, New York.

Hodes, L., and E. Specker 1968. "Length of Formulas and Elimination of Quantifiers," in Contributions to Mathematical Logic, K. Schutte (ed.), pp. 175-188. North Holland, Amsterdam.

Hopcroft, J., W. Paul, and L. Valiant 1977. "On Time Versus Space," J. Assoc. Computing Machinery. Vol. 24, pp. 332-337

Karnaugh, M. 1965. "The Map Method for Synthesis of Combinational Logic Circuits," *Trans. AIEE*, Pt. 1., Vol. 72, pp. 593-599.

Karp, R.M. 1972. "Reducibility among Combinatorial Problems," in *Complexity of Computer Computations*, ed. R.E. Miller and J. Thatcher, pp. 85-104. Plenum Press, New York.

Khasin, L.S. 1970. "Complexity Bounds for the Realization of Monotonic Symmetrical Functions by Means of Formulas in the Basis $\vee, \wedge, \neg$," *Dokl. Akad. Nauk SSSR*, Vol. 189, pp. 752-755 (English Translation in *Soviet Physics-Doklady*, Vol. 14, pp. 1149-1151).

Khrapchenko, V.M. 1971. "Complexity of the Realization of a Linear Function in the lass of π-Circuits," *Matematichskie Zametki*, Vol. 9, pp. 35-40 (English Translation in *Matematics Notes of the Academy of Sciences of the USSR*, pp. 21-23).

Kleiman, Mark and Nicholas Pippenger 1978. "An Explicit Construction of Short Monotone Formulae for the Monotone Symmetric Functions," *Theoretical Computer Science*, to appear.

Kleitman, D. and G. Markowsky 1975. "On Dedekind's Problem: The Number of Isotone Boolean Functions, II," *Transactions of the American Mathematical Society*, Vol. 213, pp. 373-390.

Knuth, Donald E. 1968. *The Art of Computer Programming: Volume 1 - Fundamental Algorithms*. Addison-Wesley, Reading, Massachusetts.

Knuth, Donald E. 1973. *The Art of Computer Programming: Volume 3 - Sorting and Searching*. Addison-Wesley, Reading, Massachusetts.

Krichevskii, R.E. 1961. "Realization of Functions by Superposition," *Probl. Kibern*, Vol. 2, pp. 123-238 (English Translation in *Problems of Cybernetics*, Vol. 2, pp. 458-477).

Krichevskii, R.E. 1964. "Complexity of Contact Circuits Realizing a Function of Logical Algebra," *Dokl. Akad. Nauk SSSR*, Vol. 151, pp. 803-806 (English Translation in *Soviet Physics - Doklady*, Vol. 8, pp. 770-772).

Lamagna, E.A. 1975. *The Complexity of Monotone Functions*. Technical Report CS-3, Computer Science Program and Division of Applied Mathematics, Brown University, Providence, Rhode Island.

Lamagna, E.A., and J.E. Savage 1974. "Combinational Complexity of Some Monotone Functions," *Proceedings of the Fifteenth IEEE Symposium on Switching and Automata Theory*, pp. 140-144.

Lipton, R.J., and R.E. Tarjan, 1977. "Applications of a Planar Separator Theorem," *Proceedings of the Eighteenth IEEE Symposium on Foundations of Computer Science*, pp. 162-170.

Lupanov, O.B. 1958. "A Method of Circuit Synthesis," *Izvestia VUZ (Radiofizika)*, No. 1, pp. 120-140.

Lupanov, O.B. 1962. "Complexity of Formula Realization of Functions in Logical Algebra," *Probl. Kibern*, Vol. 3 (1960), pp. 61-80 (English Translation in *Problems of Cybernetics*, Vol. 3, pp. 782-811).

MacLane, Saunders, and Garrett Birkoff 1967. *Algebra*. The MacMillan Company, New York.

McColl, W.F. 1977. *Some Results on Circuit Depth*. University of Warwick Theory of Computation Report 18, Department of Computer Science, Coventry, England.

McColl, W.F. and M.S. Paterson 1977. "The Depth of All Boolean Functions," *SIAM Journal on Computing*, Vol. 6, pp. 373-380

McCluskey, E.J., Jr. 1956. "Minimization of Boolean Functions," *Bell System Technical Journal*, Vol. 35, pp. 1417-1444.

Mehlhorn, K. and Z. Galil 1976. "Monotone Switching Circuits and Boolean Matrix Product," *Computing*, Vol. 16, pp. 99-111.

Muller, D.E. 1956. "Complexity in Electronic Switching Circuits," *IRE Transactions on Electronic Computers*, Vol. EC-5, pp. 15-19

Muller, D.E. and F.P. Preparata 1975. "Bounds to Complexities of Networks for Sorting and for Switching," *J. Assoc. Computing Mach.*, Vol. 22, pp. 195-201.

Neciporuk, E.I. 1966. "A Boolean Function," *Dokl. Akad. Nauk SSSR*, Vol. 169, pp. 765-766 (English Translation in *Soviet Math. Doklady,* Vol. 7, pp. 999-1000).

Neciporuk, E.I. 1971. "On a Boolean Matrix," *Probl. Kibern.*, Vol. 21 (1969), pp. 237-240 (English Translation in *Syst. Theory Research*, Vol. 21, pp. 236-239).

Pan, V. Ya. 1978. "Strassen's Algorithm is not Optimal," *Proceedings of the Nineteenth IEEE Symposium on Foundations of Computer Science*, pp. 166-176.

Paterson, M.S. 1975. "Complexity of Monotone Networks for Boolean Matrix Product," *Theoretical Computer Science*, Vol. 1, pp. 13-20

Paterson, M.S. and L.G. Valiant 1976. "Circuit Size is Non-Linear in Depth," *Theoretical Computer Science*, Vol. 2, pp. 397-340.

Paul, W.J. 1976. "Realizing Boolean Functions on Disjoint Sets of Variables," Theoretical Computer Science, Vol. 2, pp. 383-396.

Paul, W.J. 1977. "A 2.5 n-Lower Bound on the Combinational Complexity of Boolean Functions," SIAM Journal on Computing, Vol. 6, pp. 427-444.

Peterson, G. 1978. An Upper Bound on the Size of Formulae for Symmetric Boolean Functions, Technical Report 78-03-01, Department of Computer Science, University of Washington, Seattle, Washington.

Pippenger, N. 1974. Short Formulae for Symmetric Functions, Report RC 5143, I.B.M. Thomas J. Watson Research Center, Yorktown Heights, New York.

Pippenger, N. 1975. "Short Monotone Formulae for Threshold Functions," unpublished manuscript.

Pippenger, N. 1976. "The Realization of Monotone Boolean Functions," Proceedings of the Eight Annual ACM Symposium on Theory of Computing, pp. 204-210.

Pippenger, N., and M.J. Fischer 1977. "Relationships Among Complexity Measures," Report RC 6569, I.B.M. Thomas J. Watson Research Center, Yorktown Heights, NY.

Post, E.L. 1941. "Two-valued Iterative Systems of Mathematical Logic," Annals of Math. Studies, Vol. 5. Princeton University Press, Princeton, New Jersey.

Pratt, V.R. 1974. "The Power of Negative Thinking in Multiplying Boolean Matrices," Proceedings of the Sixth Annual ACM Symposium on Theory of Computing, pp. 80-83.

Pratt, V.R. 1975. "The Effect of Basis on Size of Boolean Expressions," Proceedings of the Sixteenth Annual IEEE Symposium on Foundations of Computer Science, pp. 119-121.

Quine, W.V. 1952. "The Problem of Simplifying Truth Functions," American Math. Monthly, Vol. 59, pp. 521-531.

Quine, W.V. 1955. "A Way to Simplify Truth Functions," American Math. Monthly, Vol. 62, pp. 627-631.

Redkin, N.P. 1973. "Proof of Minimality of Circuits Consisting of Functional Elements," Probl. Kibern., Vol. 23 (1973), pp. 83-102 (English Translation in Systems Theory Research, Vol. 23, pp. 85-103).

Savage, J. 1976. The Complexity of Computing. John Wiley and Sons, New York.

Schnorr, C.P. 1974. "Zwei linear untere Schranken fur die Komplexitat Boolescher Functionen," Computing, Vol. 13, pp. 155-171.

Schnorr, C.P. 1976a. "A Lower Bound on the Number of Additions in Monotone Computations," Theoretical Computer Science, Vol. 2, pp. 305-315.

Schnorr, C.P. 1976b. "The Network Complexity and the Breadth of Boolean Functions," Oxford Logic Colloquium.

Schnorr, C.P. 1976c. "The Combinational Complexity of Equivalence," Theoretical Computer Science, Vol. 1, pp. 289-295.

Schnorr, C.P. 1976d. "The Network Complexity and the Turing Machine Complexity of Finite Functions," ACTA Informatica, Vol. 7, pp. 95-107.

Schnorr, C.P. 1978a. "An Algorithm for Transitive Closure with Linear Expected Time," SIAM Journal on Computing, Vol. 7, pp. 127-133.

Schnorr, C.P. 1978b. "A 3n-Lower Bound on the Network Complexity of Boolean Functions," unpublished manuscript.

Shannon, C.E. 1949. "The Synthesis of Two-Terminal Switching Circuits," Bell System Technical Journal, Vol. 28, pp. 59-98.

Spira, P.M. 1971. "On the Time Necessary to Compute Switching Functions," IEEE Transactions on Computers, Vol. C-20, pp. 104-105.

Spira, P.M. 1973. "A New Algorithm for Finding Shortest Paths in a Graph of Positive Arcs in Average Time $O(n^2 \log^2 n)$," SIAM Journal on Computing, Vol. 2, pp. 28-32.

Stockmeyer, L.J. 1974. The Complexity of Decision Procedures in Automata Theory and Logic. Lab. for Computer Science Technical Report MAC-TR-133, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Stockmeyer, L.J. 1977. "On The Combinatorial Complexity of Certain Symmetric Boolean Functions," Mathematical Systems Theory, Vol. 10, pp. 323-336.

Tarjan, R.E. 1976. "Complexity of Monotone Networks for Computing Conjunctions," unpublished manuscript.

Vilfan, B. 1972. The Complexity of Finite Functions. Project MAC Technical Report TR-97, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Wegener, I. 1978. "Switching Functions whose Monotone Complexity is Nearly Quadratic," Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, pp. 143-149.

Yao, A.C. 1974. "Bounds on Selection Networks," Proceedings of the Fifteenth Annual IEEE Symposium on Switching and Automata Theory, pp. 110-116.

Yao, A.C., D.M. Avis, and R.L. Rivest 1977. "An $\Omega(n^2 \log n)$ Lower Bound to the Shortest Paths Problem," Ninth Annual ACM Symposium on Theory of Computing, pp. 11-17. Also see correction in Tenth Annual ACM Symposium on Theory of Computing.

Yao, A.C., and R.L. Rivest 1978. On The Polynedral Decision Problem, unpublished manuscript.

Yao, F.F. Complexity of Monotone Computation of the Second Threshold Function, unpublished manuscript.

Yao, F.F. and P. Bloniarz. On the Monotone Complexity of the function threshold 2, in preparation.

Zaslavskii, I.D. "On Some Models of Computability of Boolean Functions," in Lecture Notes in Computer Science, Goos and Hartmanis (eds.), No. 32-Mathematical Foundations, pp. 153-159.

BIOGRAPHY

Peter Bloniarz was born in Boston in 1947, and spent most of his youth in Springfield, Massachusetts. He attended Cathedral High School there, and graduated from Holy Cross College in 1969. He spent two years at the University of Pennsylvania School of Medicine before receiving an M.S. degree in mathematics from the University of Massachusetts at Amherst. At M.I.T., in addition to being a student, he spent some time organizing athletic activities in the Laboratory for Computer Science. He received his Ph.D. in Computer Science in 1977 from M.I.T. He and his wife Mary Ellen have been married for eight years and have a two year old daughter Katie. They now live in Albany, New York where he teaches at SUNY-Albany.