MIT/LCS/TR–381

Cryptology and VLSI (a two-part dissertation):

I. Detecting and Exploiting Algebraic Weaknesses in Cryptosystems
II. Algorithms for Placing Modules on a Custom VLSI Chip

Alan T. Sherman
MIT Laboratory for Computer Science
Cambridge, MA 02139
October 1986

Support for this research was provided in part by the National Science Foundation (NSF) under contract number MCS–8006938, by the Defense Advanced Research Project Agency (DARPA) of the Department of Defense under contract number N00014–80–C–0622, and by the U.S. Air Force under contract number AFOSR–F49620–81–0054.

Cryptology and VLSI (a two-part dissertation):
I. Detecting and Exploiting Algebraic Weaknesses in Cryptosystems
II. Algorithms for Placing Modules on a Custom VLSI Chip

by

Alan Theodore Sherman

Sc.B., Mathematics, *magna cum laude*, June 1978
Brown University

S.M., Electrical Engineering and Computer Science, June 1981
Massachusetts Institute of Technology

SUBMITTED TO THE DEPARTMENT OF
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE
DEGREE OF

DOCTOR OF PHILOSOPHY
IN COMPUTER SCIENCE

at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
February 1987

Signature of Author_____
Department of Electrical Engineering and Computer Science
October 15, 1986

Certified by_____
Ronald Linn Rivest
Thesis Supervisor

Accepted by_____
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

Cryptology and VLSI (a two-part dissertation):

I. Detecting and Exploiting Algebraic Weaknesses in Cryptosystems
II. Algorithms for Placing Modules on a Custom VLSI Chip

Alan Theodore Sherman[1]

Submitted on October 15, 1986, to the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science.

Abstract

This dissertation describes two separate and independent investigations in cryptology and VLSI. Part I explores relationships between algebraic and security properties of cryptosystems, focusing on finite, deterministic cryptosystems whose encryption transformations form a group under functional composition. Part II explores the problem of automatically placing modules on a custom VLSI chip, focusing on the placement heuristics used in the MIT PI (Placement and Interconnect) System.

Part I. The Data Encryption Standard (DES) defines an indexed set of permutations acting on the message space $\mathcal{M} = \{0,1\}^{64}$. If this set of permutations were closed under functional composition, then the two most popular proposals for strengthening DES through multiple encryption would be equivalent to single encryption. Moreover, DES would be vulnerable to a known-plaintext attack that runs in 2^{28} steps on the average. It is unknown in the open literature whether or not DES has this weakness.

Two statistical tests are presented for determining if an indexed set of permutations acting on a finite message space forms a group under functional composition. The first test is a "meet-in-the-middle" algorithm which uses $O(\sqrt{K})$ time and space, where K is the size of the key space. The second test, a novel cycling algorithm, uses the same amount of time but only a small constant amount of space. Each test yields a known-plaintext attack against any finite, deterministic cryptosystem that generates a small group.

The cycling test takes a pseudo-random walk in the message space until a cycle is detected. For each step of the pseudo-random walk, the previous ciphertext is encrypted under a key chosen by a pseudo-random function of the previous ciphertext. Results of the test are asymmetrical: long cycles are overwhelming evidence that the set of permutations is not a group; short cycles are strong evidence that the set of

[1]Author's address: MIT Laboratory for Computer Science; 545 Technology Square; Cambridge, MA 02139.

permutations has a structure different from that expected from a set of randomly chosen permutations.

Using a combination of software and special-purpose hardware, the cycling test was applied to DES. Experiments show, with overwhelming confidence, that DES is not a group. Additional tests confirm that DES is free of certain other gross algebraic weaknesses. But one experiment discovered fixed points of the so-called "weak-key" transformations, thereby revealing a previously unpublished additional weakness of the weak keys.

Part II. The PI System is a fully automatic system for laying out custom VLSI chips. PI decomposes the layout process into separate placement, routing, and compaction phases, each of which are further decomposed into subproblems and solved by specialized component algorithms. A novel crossing placement step breaks the signal routing task into independent, fixed sized, switch-box channel routing problems.

Given a list of arbitrarily shaped rectangular modules and a list of nets which specify how the modules are to be interconnected, PI first finds a nonoverlapping layout of the modules on a rectangular surface. Modules may be flipped and rotated, but must be aligned with the edges of the chip. PI's placement algorithms attempt to minimize total chip area and the amount of wire needed for routing, while leaving enough space for routing. Subsequent routing and resizing phases complete the layout.

PI's placement strategy is built around a framework in which approximate and partial placements can be represented and manipulated. Within this general framework, PI first computes a placement hierarchy using a top-down recursive mincut-cut algorithm and then refines this initial approximate placement into an exact placement. To compute the exact placement, PI traverses the placement hierarchy in postorder, orienting the modules and determining how modules should be placed relative to each other. A data structure called the placement tree supports this process.

Key Words and Phrases

General Terms: Algorithms, combinatorial optimization, complexity theory, cryptanalysis, cryptography, cryptology, theory of computation, very large scale integration (VLSI).

Specific Terms: Channel definition, channel routing, closed cipher, compaction, crossing placement, custom VLSI, cycle detection algorithm, Data Encryption Standard (DES), finite permutation group, global routing, graph partitioning, group detection game, idempotent cryptosystem, layout algorithm, mincut, multiple encryption, pad placement, PI Project, PI System, placement algorithm, pure cipher, RSA cryptosystem, weak keys.

Thesis Supervisor: Ronald Linn Rivest
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I am deeply grateful to my thesis advisor, Professor Ronald Linn Rivest, for several years of helpful guidance, supervision, and encouragement. It has been an enriching experience to work under such an outstanding researcher, educator, and thoughtful human being.

I thank Professors Shafi Goldwasser, Charles E. Leiserson, and Silvio Micali for their comments and advice as thesis readers. I would also like to express my appreciation to Burton S. Kaliski for his remarks on preliminary drafts of my work.

Several of the algebraic tests described in the first part of this thesis have been applied to the Data Encryption Standard (DES) using a combination of software and special-purpose hardware. I am especially grateful to Burton Kaliski for his contribution to this experimental work. Burt performed a herculean role in designing, building, and testing the special-purpose hardware. Leon Roisenberg assisted Burt in building the hardware, and supporting software was written by John Hinsdale, Kaliski, and Rivest. I would like to thank the Functional Languages and Architectures (FLA) Research Group at the MIT Laboratory for Computer Science for letting us use their hardware laboratory. My thanks also go to the International Business Machines Corporation (IBM) for donating an IBM Personal Computer which served as host to the special-purpose hardware.

Since part of my thesis deals with my work on the PI System, I would like to acknowledge the other students who also participated in the design and development of PI. The major contributors were Alan Baratz, Arthur Chin, Chee-Seng Chow, David Christman, Ali Ghaznavi, Alain Hanover, David Hsu, David Jilk, Joe Kilian, Jim Koschella, Michael Koss, Gordon Linoff, Andrew Moulton, Mark Novick, Ron Pinter, Flavio Rose, and Susmita Sur. The PI Project reflects the combined work of these people.

Through their interest in PI, and through their use and modifications of PI, the General Electric Research Center (GE) in Schenectady, New York, also contributed to the PI project. I thank Robert M. Mattheyses and Ross Stenstrom for sharing their thoughts and experiences with using PI at GE.

I am deeply grateful to Tomoko Shimakawa for her love and support, and for drawing all of the thesis figures.

Ray Hirschfeld built a large version of LaTeX, making it possible for me to compile this document.

My thanks also go to László Babai, Robert W. Baldwin, Don Coppersmith, Gary Miller, and Adi Shamir for helpful comments.

Finally, I would like to thank the organizations that supported me financially during my graduate studies. Most of my financial support came in the form of research and teaching assistantships. The MIT Laboratory for Computer Science provided my research assistantships, which were sponsored in part by the National

Science Foundation (NSF) under contract number MCS–8006938, by the Defense Advanced Research Project Agency (DARPA) of the Department of Defense under contract number N00014–80–C–0622, and by the U.S. Air Force under contract number AFOSR–F49620–81–0054. These grants also supported the PI Project. I carried out my teaching assistantships in the MIT Department of Electrical Engineering and Computer Science, where I helped teach courses in algorithms and introductory computer science. In addition, the General Electric Foundation/Ford Foundation awarded me several forgivable loans. I appreciate all of this generous support.

Contents

This document was produced at the MIT Laboratory for Computer Science using the EMACS text editor, the LaTeX document preparation system, and an Imagen laser-driven xerographic printer.

List of Figures

List of Tables

Chapter 1

Introduction and Overview

Over the next decades, cryptology and VLSI will play an increasingly important role in our lives. As people make greater use of computers to conduct business transactions and to store and communicate confidential information, we will depend more on cryptography to safeguard the privacy, authentication, and integrity of digital information. As VLSI technology makes it possible to build miniature silicon chips cheaply and reliably, more products—from dishwashers to space ships—will exploit the power of the microchip. Already, many people enjoy the harmonious combination of cryptology and VLSI through automatic teller machines and smart credit cards.

Cryptology and VLSI are appealing not only for their important practical applications, but also for the deep issues they raise in understanding what makes computation easy and what makes computation hard. Cryptographers attempt to create problems that are infeasible to solve for anyone who does not possess the secret key, and cryptanalysts strive to defeat the cryptographers. VLSI chip designers attempt to exploit the power of parallel computation, but they must first confront the difficult problem of laying out their chips efficiently.

This dissertation describes two separate and independent investigations in cryptology and VLSI. Part I explores relationships between algebraic and security properties of cryptosystems; part II addresses the problem of placing modules on a custom VLSI chip. Each part grapples with practical issues using a blend of theoretical and experimental approaches. For example, part I proposes new algorithms for detecting and exploiting algebraic structure in cryptosystems and applies some of these algorithms to the Data Encryption Standard (DES).[1] Similarly, part II describes new methods for laying out custom VLSI chips automatically and discusses how these ideas were implemented in the PI (Placement and Interconnect) System.[2] Although

[1] DES is a federal standard for the cryptographic protection of computer data, adopted in 1976 by the United States National Bureau of Standards.

[2] The PI System is an automatic layout system for custom VLSI, developed at MIT under the leadership of Professor Ronald Rivest.

there are many important relationships between cryptology and VLSI, and although these relationships were partially responsible for my initial interest in VLSI, this dissertation makes no attempt to explore these relationships.

Organized in three sections, the rest of this chapter introduces and outlines each part of the dissertation and summarizes the main research contributions. Additional introductory material of a more detailed and technical nature is found in chapters 2 and 8.

1.1 Cryptology

Cryptology is the science of making and breaking codes and ciphers.[3] Although cryptology has been and continues to be of great interest to governments, military forces, and amateur mathematicians, today cryptology is becoming increasingly more important in the public sector.[4] As we depend more and more on computer networks and information systems, the need to protect digital information and computer transactions grows. Without cryptographic protection, grave dangers exist for fraud and invasion of privacy.

Communications security deals with a variety of problems associated with the safe transmission of information among parties in the presence of adversaries. Three of the most important problems of communications security are the problems of secrecy, authentication, and integrity. The *secrecy* problem is to ensure that a message can be read only by the intended receiver. The *authentication* problem is to provide assurance to the intended receiver of a message that the received message actually came from the claimed sender. The *integrity* problem is to guarantee that an adversary cannot modify messages without being detected.

Conventional cryptography works by taking advantage of a powerful asymmetry: the sender and receiver share some secret information, called the *key*, that that is unknown to the enemy. During encryption, the message and the key are thoroughly mixed in a complicated way. If the encryption process is secure, only someone who knows the key can unscramble the message. Two crucial ingredients of this process are the random selection of the key and the mixing operation that is hard to reverse.

In *public-key cryptography*, a different asymmetry exists [154]. Here, separate keys are used for encryption and decryption. Only the receiver holds the secret (decryption) key. Everyone else, including the adversary, has access only to a corresponding public (encryption) key. Anyone can encrypt, but only someone who knows the secret key can decrypt. Although public-key cryptography avoids the need for the

[3] Strictly speaking, *cryptography* is the field of making codes and ciphers, and *cryptanalysis* is the field of breaking codes and ciphers. Together cryptography and cryptanalysis comprise cryptology, which is one branch of the larger domain of *communications security*.

[4] Although communications security is studied extensively in the classified world [72], I have never had access to any classified materials.

sender and receiver to exchange secret keys, the sender must have assurance that the public key he believes corresponds to the receiver is actually the public key of the intended receiver. One important advantage of public-key cryptography over conventional cryptography is the additional capability it provides to create digital signatures [226].

Much of my interest in cryptology has focused on the question, "What does it mean, in a precise mathematical sense, for a cryptosystem to be secure?" This is a crucial question. Without an answer to this question, our understanding of cryptology is foggy and our assessments of cryptosystems are unscientific. Of course, there is no single answer to this question. Security is a relative concept which depends on many factors including the resources and capabilities of the enemy, and what it means to "break a cryptosystem." Security also depends on what one means by a "cryptosystem."

In his seminal paper on cryptography, Claude Shannon proposed an information theoretic definition of security, called *perfect secrecy* [150]. Shannon's notion of security requires that the enemy have insufficient information to decipher encrypted messages, even if given unlimited computational resources. The well-known *one-time pad* satisfies Shannon's notion of perfect secrecy.

More recently, several attempts have been made to base definitions of cryptographic strength on computational complexity [130,138,147]. Informally, these definitions say that a cryptosystem is secure if and only if breaking it requires an unreasonably large amount of time or space. Over the past decade, several computational complexity based cryptosystems have been proposed [226,252,207,217,239,246,129]. In some cases, researchers have proven that their cryptosystems satisfy particular definitions of security. However, even for such "provably secure" cryptosystems, security is conditional on unproven conjectures in computational number theory. Specifically, cryptographers have attempted to link the difficulty of breaking their cryptosystems with the difficulty of solving mathematical problems that many experts conjecture are difficult to solve. The most popular of these problems have been the integer factoring problem, the quadratic residuosity problem, the discrete logarithm problem, and the elliptic logarithm problem. Since computer scientists have not proven any nontrivial lower bound on the amount of computer resources required to solve any nontrivial problem, this approach is the most one could expect without making monumental advances in complexity theory. Thus, once again, security becomes a relative notion.

Part I of this dissertation does not attempt to give a final explanation of the meaning of cryptographic strength. Instead, part I addresses a focused dimension of this question by exploring the thesis that *there are important relationships among algebraic and security properties of cryptosystems.*

Throughout part I attention is restricted to finite, deterministic cryptosystems that consist of an indexed set of permutations acting on a finite message space.

Within this context, relationships are explored between algebraic properties of the set of permutations and security properties of the cryptosystem.

Outline of Part I

Part I investigates relationships among algebraic and security properties of cryptosystems, focusing on how algebraic properties can be detected and exploited in the context of finite deterministic cryptosystems. The major theoretical contributions of part I are two statistical tests and their related known-plaintext attacks for group ciphers. The major practical contributions are the application of one of these tests and other algebraic tests to DES. With one exception, our experimental results are consistent with the hypothesis that DES acts like a set of randomly chosen permutations. But one experiment detected fixed points for the so-called "weak key" transformations, thereby discovering a previously unpublished additional weakness of the weak keys.

Listed below are brief summaries of the six chapters that constitute part I.

- *Chapter 2* illustrates the main ideas of part I through three examples. First, the chapter explains how the cycling closure test can be carried out on DES. Second, the chapter explains how the meet-in-the-middle attack against group ciphers could be applied against DES, if DES were a group. Third, the chapter examines relationships among algebraic and security properties in the RSA cryptosystem.

- *Chapter 3* presents an assortment of background information helpful in understanding the rest of part I. This chapter describes basic properties of finite deterministic cryptosystems, reviews some previous work on DES, and explains the notation and terminology used throughout part I.

- *Chapter 4* presents a meet-in-the-middle test and a constant space cycling test for for determining if a finite deterministic cryptosystem forms a group under functional composition. The chapter also describes several other related statistical tests for detecting algebraic structure in cryptosystems.

- *Chapter 5* explains how each of the two main tests from the previous chapter can be transformed into a known-plaintext attack against group ciphers.

- *Chapter 6* describes experimental work in which we performed the cycling closure test and other algebraic tests on DES. The chapter explains our findings, lists detailed experimental results, and describes special-purpose hardware used in to carry out the tests.

- *Chapter 7* suggests two directions for further research. This chapter lists several open questions about DES and states as a two-person game the problem of determining whether or not a set of cryptographic transformations forms a group.

1.2 VLSI

Large Scale Integration (LSI) and *Very Large Scale Integration (VLSI)* refer respectively to the high and very high densities of wires and active components that can be placed on a silicon chip. Although these terms have no widely agreed-upon precise meanings, today, when someone refers to a chip as "VLSI," the chip might have as many as several hundred thousand transistors in a region about one square centimeter in area. For many circuits, increasing integration density decreases chip size, increases chip speed, and decreases chip cost. Thus, VLSI helps build smaller, faster, and cheaper electronic devices.

A *chip* is a slice of material that implements an electronic circuit, where an *electronic circuit* consists of *active components* (*e.g.* transistors) electrically connected by *wires*. There are several different technologies for building chips. Among these technologies is the popular *metal-oxide-semiconductor (MOS)* technology. In three-layer MOS, a chip consists of three layers of electrically conducting material which are separated by insulating matter. The conducting layers are made of *metal*, *polysilicon*, and *diffusion*. The circuit is realized by etching paths in each of the three layers. Paths in metal, polysilicon, or diffusion form wires; a transistor is created whenever a polysilicon path crosses a diffusion path.[5]

A *custom chip*—as opposed, for example, to a *semicustom chip*—is a chip whose active components are not restricted to be laid out in any regular pattern. With custom VLSI, it is especially convenient to build a chip by putting together a collection of available parts.

Part II of this dissertation addresses the problem of deciding where on a custom chip the active components should be placed. Given the huge number of wires and transistors that can be fabricated on a single chip, there is a strong need to develop efficient algorithms for placing the active components. Although part II is presented in terms of custom VLSI, the results also apply to most other layout technologies.

The following layout model will be used throughout part II. Active components are represented by *modules*, which are arbitrarily sized rectangles with communication points (called *pins*) along the edges. Modules may be flipped and rotated, but must be oriented with the edges of the chip. Modules are not permitted to overlap. Two layers are available for routing, but wires are not allowed to cross modules.

[5] Actually, this simple description of how a transistor is created is only an abstraction of a more complicated process [339,341,342].

We will focus our attention on a layout methodology that separates the layout process into separate placement and routing phases. Input to the placement phase consists of a set of modules and set of *nets* which describe how the modules are to be interconnected. Output describes a placement of the modules in a rectangular region of the plane. The goal is to minimize estimated layout cost, while leaving enough room in between the modules to route the wires. Usually, we will measure layout cost by total chip area and wire length.

The major portion of part II deals with a set of placement heuristics developed for the MIT PI System. These heuristics are based on a top-down recursive mincut strategy[6] that combines geometric and graph-theoretic placement concerns.

Outline of Part II

Part II deals primarily with the placement heuristics that were designed and implemented by Alan Sherman for the PI System. The major contributions are the placement algorithms and the general framework in which these algorithms operate. Most of part II is devoted to describing PI's placement algorithms. Along the way, part II identifies layout issues as seen through PI and explains how PI deals with these issues. Several ideas for extending PI's placement algorithms are presented. In addition, part II formulates several theoretical problems abstracted from and motivated by PI. Part II concludes with an analysis of the PI Project.

Listed below are brief summaries of the eight chapters that constitute part II:

- *Chapter 8* gives a brief overview of the PI System. Pictures taken from a computer terminal illustrate PI's performance on a small example.

- *Chapter 9* describes the PI System in more detail, concentrating on its objectives, layout model, and major design decisions. This chapter also defines the notation and terminology used throughout part II.

- *Chapter 10* summarizes PI's routing and resizing algorithms. This chapter is included primarily for the reader who would like to learn more about PI. Although the rest of part II does not depend on chapter 10, the reader may find this chapter helpful in better understanding how PI's placement and routing algorithms interact.

- *Chapter 11* introduces PI's placement algorithms. This chapter explains the framework in which PI's placement algorithms operate and summarizes the particular algorithms that PI uses within this framework.

[6] A *mincut heuristic* is a heuristic that involves partitioning a graph into two sets of vertices such that the number of edges that have endpoints in both sets is made as small as possible.

- *Chapter 12* explains each of PI's placement algorithms in detail. Specifically, this section explains how PI uses a top-down mincut heuristic to find an approximate placement of the modules, how PI adjusts an approximate placement by orienting (flipping and rotating) the modules, and how PI uses a bottom-up "hardening" procedure to transform any approximate placement produced by the mincut heuristic into an exact, legal placement.

- *Chapter 13* describes several extensions to the PI System that were considered for PI, but which were never implemented. These extensions include bottom-up pairing and combinatorial search placement heuristics, a new crossing placement algorithm, and ideas for additional channel routers. The chapter also describes an extension to PI developed and used by the General Electric Company.

- *Chapter 14* formulates several theoretical placement problems motivated by and abstracted from PI. Most of these problems remain as open research questions.

- *Chapter 15* summarizes the major contributions of the PI Project and evaluates how well PI met its original objectives. This chapter also discusses several other miscellaneous topics relating to PI, including the history and implementation of PI, and the current status of the PI Project.

1.3 Research Contributions

This section briefly summarizes the major research contributions of this dissertation.

The major results of part I deal with cryptosystems that are groups. Two attacks against group ciphers are presented, and a novel cycling test is described for determining if a cryptosystem is a group.

Part I also presents experimental work in which statistical algebraic tests were applied to the DES using a combination of software and special-purpose hardware. Experimental evidence shows, with overwhelming confidence, that DES is not a group. Additional tests show that DES is free of certain other gross algebraic weaknesses. But one experiment unexpectedly discovered fixed-points for the so-called "weak key" transformations, thereby revealing an additional weakness of the weak keys previously unknown in the open literature.

The major contributions of part II deal with the design and implementation of the PI System, and especially with the PI System's heuristics for placing modules on a custom VLSI chip. Part II explains PI's placement algorithms, which were designed and implemented by Alan Sherman. Part II also presents a thorough description and analysis of the PI System, identifying layout issues as seen by PI and discussing how PI deals with these issues. An example of PI's step-by-step performance is shown.

PI's problem decomposition and placement framework provide an effective approach for laying out custom VLSI chips.

The PI System places modules in three steps. First, PI applies a top-down mincut heuristic to compute an approximate placement of the modules. Second, PI adjusts the placement by flipping and rotating the modules. Third, PI applies a bottom-up procedure to transform the approximate placement into an exact placement with space left for routing. These algorithms are built around a general framework that can support a variety of placement heuristics.

Part II also describes several extensions to the PI System and states several problems abstracted from and motivated by PI.

Portions of this dissertation present joint work with Ronald Rivest, Burton Kaliski, and members of the PI Project.

Part I

Detecting and Exploiting Algebraic Weaknesses in Cryptosystems

Chapter 2

Relationships Between Algebraic and Security Properties of Cryptosystems

Fundamental algebraic properties such as commutativity, associativity, multiplicativity, and closure play a significant role in the structure of mathematics [4]. They also play an important role in cryptology. Although some work has been done on algebraic properties and the security of cryptographic protocols [285,286], the relationship between algebraic properties and computational complexity security properties of cryptographic functions remains barely explored in the open literature. Part I begins to explore this fascinating territory.

Part I focuses primarily on the computational complexity security properties of group ciphers—ciphers whose cryptographic functions form a group under functional composition. Two known-plaintext attacks for exploiting group ciphers and two novel statistical tests for detecting group ciphers are presented. My interest in group ciphers was motivated largely by the following two reasons. First, the group is one of the most fundamental algebraic structures, and hence group ciphers are a natural context in which to investigate relationships among algebraic and security properties. Second, I was intrigued by the question of whether or not the set of DES transformations forms a group and by the fact that the common modulus variation of the RSA cryptosystem forms a group under functional composition. The study of group ciphers in part I continues a research direction initiated by Shannon, who examined information theoretic security properties of group ciphers [150].

Part I also describes experiments in which several statistical algebraic tests were applied to DES. These tests confirmed popular belief that DES is free of certain gross algebraic weaknesses. But one test discovered fixed points of the so-called "weak key" transformations, thereby revealing a previously unpublished additional weakness of the weak keys.

Algebraic structure in cryptosystems can have two-sided effects. On the one hand, algebraic structure can provide a framework that makes encryption and decryption possible. It can also endow a cryptosystem with desirable capabilities or security properties. On the other hand, algebraic structure can weaken a cryptosystem by providing the cryptanalyst with something to exploit. Sometimes, special capabilities of a cryptosystem due to algebraic structure come at a price of less efficient use of key bits to reach certain security levels. While there are general relationships between some algebraic and security properties of cryptosystems, how algebraic structure affects a cryptosystem depends in part on the details of the particular system.

The rest of this chapter introduces the main ideas from part I through illustrating how algebraic and security properties interact in two particular cryptosystems. Section 2.1 explains why it is important to know whether or not DES is a group and summarizes how we applied the cycling closure test from chapter 4 to show, with overwhelming confidence, that DES is not a group. Section 2.2 describes how the meet-in-the-middle attack from chapter 5 could be applied against DES, if DES were a group. Section 2.3 briefly discusses relationships between algebraic and security properties of the RSA cryptosystem. This section also explains how the attacks against group ciphers from chapter 5 can be applied against the RSA cryptosystem, even though these attacks turn out to be less efficient at breaking RSA than are known factoring methods.

2.1 Is the Data Encryption Standard a Group?

The Data Encryption Standard (DES) is a federal standard for the cryptographic protection of computer data and is widely used by banks and other organizations to protect unclassified data. Although a few studies on DES have been openly published,[1] to date, numerous fundamental questions about the standard remain unanswered in the open literature. Part I addresses one such important question: "Is the set of DES transformations closed under functional composition?"

DES defines an indexed set of permutations acting on the message $M = \{0,1\}^{64}$. There are $M = 2^{64}$ messages and $K = 2^{56}$ keys. Each key k represents a transformation T_k, with inverse T_k^{-1}. Let $K = \{1,0\}^{56}$ denote the set of keys.

It is important to know whether or not DES is closed since, if DES were closed, it would have the following two weaknesses. First, both sequential multiple encryption and Tuchman's multiple encryption scheme—the two most popular proposals for strengthening DES through using multiple encryption—would be equivalent to single encryption.[2] Even worse, DES would be vulnerable to a known-plaintext attack that

[1] See bibliography for a list of selected technical works on DES. For an overview of DES, see [59], [68], or [66].

[2] To encrypt a message x using *sequential multiple encryption* is to compute $T_i T_j(x)$, where the

runs in 2^{28} steps, on the average. Each weakness follows from the fact that the set of cryptographic transformations of any closed cipher forms a group under functional composition. Although most researchers believe DES is not closed, no one has proven this conjecture in the open literature.

Chapter 4 describes two statistical tests for determining whether or not an indexed set of permutations acting on a finite message forms a group under functional composition. We will now summarize how we applied one of these tests—the cycling closure test—to DES.

Let x_0 be any message and consider the set S_{x_0} recursively defined as follows: x_0 is an element of S_{x_0} and, for any key k and any message $x \in S_{x_0}$, $T_k(x)$ is also an element of S_{x_0}. Thus, S_{x_0} is the set of messages that can be reached through multiply encrypting x_0 zero or more times with arbitrary keys.

If DES acted like a set of randomly chosen permutations, then we would expect $S_{x_0} = \mathcal{M}$ and thus $|S_{x_0}| = M = 2^{64}$. However, if DES were closed, then $|S_{x_0}| \leq K = 2^{56}$, since sequential multiple encryption would be equivalent to single encryption and there are at most K distinct encryption transformations. The cycling closure test computes a statistic based on the size of S_{x_0}.

The cycling closure test picks an initial message x_0 at random and then takes a pseudo-random walk in S_{x_0}, beginning at x_0. For each step of the pseudo-random walk, the previous ciphertext is encrypted under a key chosen by a pseudo-random function of the previous ciphertext. The walk continues until a cycle is detected. By the "Birthday Paradox," the walk is expected to cycle after approximately $|S_{x_0}|^{1/2}$ steps.

More specifically, the test computes a sequence of messages $x_0, x_2, \ldots$. For each $i \geq 0$, the next message x_{i+1} is computed by

$$x_{i+1} = f_\rho(x_i) \tag{2.1}$$

where the function $f_\rho : \mathcal{M} \to \mathcal{M}$ is defined by

$$f_\rho(x) = T_{\rho(x)}(x) \tag{2.2}$$

for all messages $x \in \mathcal{M}$. The walk is guided by a deterministic, pseudo-random function $\rho : \mathcal{M} \to \mathcal{K}$ that maps messages to keys. If ρ is "random," then f_ρ acts like a random function on S_{x_0}.

Since S_{x_0} is finite, the walk will eventually encounter the same message twice. Thereafter, the walk will remain periodic because f_ρ is deterministic. Let λ be the least integer such that $x_\lambda = x_i$ for some $0 \leq i < \lambda$, and let μ be the least positive integer such that $x_{\lambda+\mu} = x_\lambda$. The walk is completely determined by the *leader* $x_0, x_1, \ldots, x_{\lambda-1}$ and the *cycle* $x_\lambda, x_{\lambda+1}, \ldots, x_{\lambda+\mu}$. The integers λ and μ are

keys i and j are chosen independently. Similarly, to encrypt a message x under *Tuchman's scheme* is to compute $T_i T_j^{-1} T_k(x)$, where the keys i, j, and k are independently chosen [197,68,307].

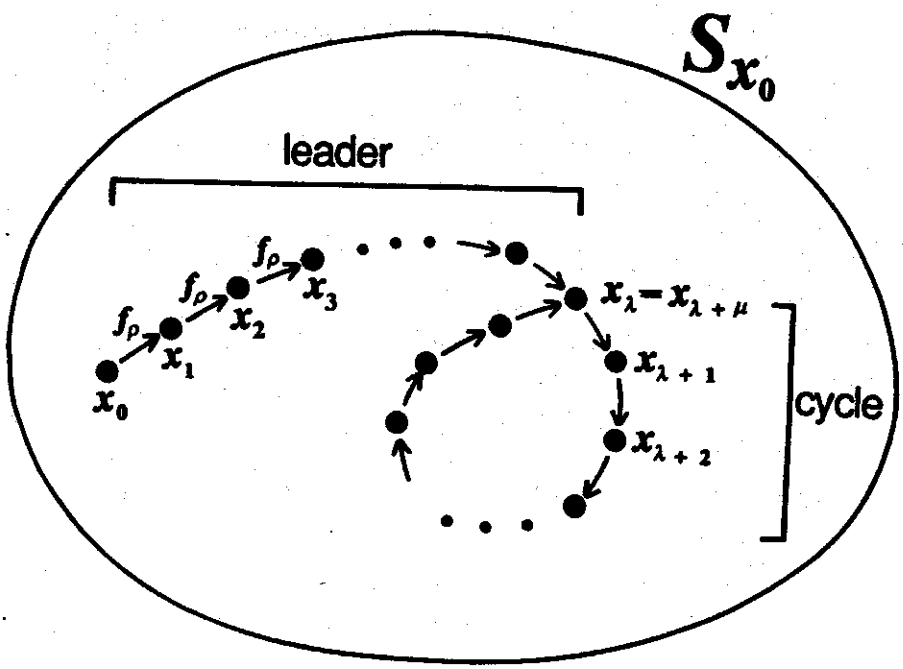


Figure 2.1: The cycling closure test takes a pseudo-random walk in the message space

called respectively the ***leader length*** and ***cycle length*** of the sequence $x_0, x_1, \ldots$. See figure 2.1.

Results of the test are asymmetrical. Walks significantly longer than $\sqrt{K} = 2^{28}$ are strong evidence that DES is not a group. Walks significantly shorter than $\sqrt{M} = 2^{32}$ are strong evidence that DES has a structure different from that expected from a set of randomly chosen permutations.

Using a combination of software and special-purpose hardware, we carried out this cycling test on DES. To detect cycles and to compute cycle and leader lengths, we used a variation of the Sedgewick-Szymanski cycle detection algorithm [97,98,99]. Our experiments detected cycles after approximately 2^{33} steps, giving overwhelming evidence that DES is not a group (see chapter 6).

2.2 Meet-in-the-Middle Attack Against Group Ciphers

All group ciphers are vulnerable to known-plaintext attacks that run in time $O(\sqrt{K})$, where K is the size of the key space. Chapter 5 presents two such attacks. We will now illustrate one of these attacks by explaining how it could be applied against DES, if DES were a group. The attack we will describe is called the ***meet-in-the-middle attack.***

As in the previous section, let T_k denote DES encryption under key k. Input to the attack consists a short sequence of matched plaintext/ciphertext pairs $(p_1, c_1), (p_2, c_2), \ldots, (p_l, c_l)$ each derived from the same key k. With high probability, the attack finds a representation of the encryption function as a product $T_k = T_b T_a$. The cryptanalyst can use this representation of T_k to encrypt and decrypt additional messages. This attack does not find the key k.

Let $p = p_1$ and $c = c_1$. To find a pair of keys a, b such that $T_k = T_b T_a$, the cryptanalyst computes $x_i = T_{a_i}(p)$ and $y_j = T_{b_j}^{-1}(c)$ for all $1 \leq i, j \leq r$ for $2r$ randomly chosen keys $a_1, a_2, \ldots, a_r$ and $b_1, b_2, \ldots, b_r$. The cryptanalyst searches for a "match" $x_i = y_j$ and then verifies that $T_k = T_b T_a$ using the additional plaintext/ciphertext pairs. Matches can be found by sorting the triples $(x_i, a_i, \text{"}A\text{"})$ and $(y_j, b_j, \text{"}B\text{"})$, for $1 \leq i, j \leq r$, on their first components. See figure 2.2.

If the set of DES encryption transformations formed a group under functional composition, then as b_j varies over all possible keys, $T_{b_j}^{-1} T_k$ would range over all encryption transformations. Consequently, by the "Birthday Paradox" (see section 3.3), the expected number of keys that would have to be tried until a match is found is approximately $\sqrt{K} = 2^{28}$.

The meet-in-the-middle attack requires $2r$ encryptions, $O(r)$ words of memory, plus the time to generate the keys and to look for matches. By choosing $r = c\sqrt{K}$ for some sufficiently large constant c, the chance of finding a match can be made as

$$D_{d,n}(y) = y^d \bmod n \tag{2.4}$$

for all $x, y \in Z_n$. By Euler's generalization of Fermat's little theorem, it can be easily shown that $D_{d,n}(E_{e,n}(x)) = x$ for all messages $x \in Z_n$. The requirement that the exponent e have a multiplicative inverse modulo $\phi(n)$ ensures that encryption is bijective.

When RSA is used as a public-key cryptosystem, the public key is the pair (e, n); the secret key is d. The primes p and q must also be kept secret, since d can be quickly computed from n, e, p, and q. Although proposed as a public-key cryptosystem, RSA can also serve as a conventional cryptosystem. When used as a conventional cryptosystem, the key consists of the triple (e, d, n).

The security of RSA rests in part on the difficulty of factoring the modulus n: given the factors p and q of n, an enemy can quickly compute the secret key d. Although several attacks against RSA have been shown equivalent to factoring n [226,232,229,222,230], no one has proven that breaking RSA necessarily implies ability to factor n. Thus, breaking RSA is at most as difficult as factoring n.[4] The best known techniques for factoring n run in time $O(L(n))$, where $L(n) = e^{\sqrt{\log n \log\log n}}$ [107,105,103].

The RSA scheme actually defines a *uniform family of cryptosystems* where each n determines one cryptosystem in the family. The family is uniform in that each cryptosystem in the family is "essentially the same," except for the parameter n. By choosing n appropriately, various levels of security can be achieved. The quantity $s = \log_2 n$ is sometimes referred to as the *security parameter* of the system.

Algebraic Properties of RSA

The RSA cryptosystem is rich with algebraic structure. To begin with, the message subspace Z_n^* forms a group under multiplication modulo n. Moreover, RSA has the following six algebraic properties:

1. *Multiplicativity.* For every key (e, n), for all messages x, y, it is true that $E_{e,n}(x)E_{e,n}(y) \bmod n = E_{e,n}(xy \bmod n)$.

2. *Complementation property.* For every key (e, n), for every message x, it is true that $E_{e,n}(n - x) = n - E_{e,n}(x)$.

3. *Commutativity.* For every modulus n, for every encryption exponents e_1, e_2, it is true that $E_{e_1,n}E_{e_2,n} = E_{e_2,n}E_{e_1,n}$.

[4] Note, however, that breaking a related cryptosystem due to Williams [233] and Rabin [225] is as hard as factoring the modulus. In William's scheme, the encryption exponent e is always taken to be 2, yielding a 4-to-1 encryption transformation. But William's scheme has some weaknesses, including susceptibility to a chosen-ciphertext attack [143].

4. *Homogeneity.* For every decryption key (d, n), there exists an encryption key (e, n), such that $D_{d,n} = E_{e,n}$.

5. *Existence of fixed points.* For every modulus n, there are at least nine "fixed point" messages x such that, for every encryption exponent e, $E_{e,n}(x) = x$. Among these messages are $n-1, 0, 1, p$, and q [214].

6. *Preservation of quadratic character.* For every message x, for every key (e, n), it is true that x is a quadratic residue module n if and only if $E_{e,n}(x)$ is a quadratic residue module n.

Most of these properties follow from the fact that that each RSA encryption function is a group homomorphism of the multiplicative group modulo n.

Discussion

The algebraic structure of RSA give this cryptosystem several important functional capabilities and security properties.

For example, the commutativity and homogeneity properties imply that encryption and decryption commute; that is, $D_{d,n}(E_{e,n}(x)) = E_{e,n}(D_{d,n}(x))$ for all keys e, d, n and every message x. Because encryption and decryption commute, $E_{e,n}(D_{d,n}(x)) = x$ for all messages $x \in Z_n$, and hence RSA can be used as a digital signature scheme.

By the multiplicativity property, $D_{d,n}(y) = D_{d,n}(yr^e)r^{-1} \bmod n$ for any $y, r \in Z_n$. Hence, to decrypt any given ciphertext $y \in Z_n$, it suffices to decrypt yr^e for any $r \in Z_n$. This fact gives RSA a *uniform security* property which says informally that, if RSA is hard to break on some small fraction of the message space, then RSA is hard to break almost everywhere. More specifically, for any $0 \leq \epsilon < 1$, if there exists a polynomial time algorithm (polynomial in $\log_2 n$) for computing e-th roots modulo n for a fraction ϵ of the messages in Z_n^*, then there exists a probabilistic polynomial time algorithm (polynomial in $\log_2 n$ and $1/\epsilon$) for computing e-th roots for all messages in Z_n^* [270].

As shown by Chor and others [216,217,248], the multiplicativity property together with the complementation property gives RSA a double-edged *bit security* property: given any RSA ciphertext, computing the least significant bits of the plaintext is as hard (*i.e.* polynomial-time equivalent) as computing the entire plaintext.

But the multiplicativity property also creates dangers in situations that permit chosen-ciphertext attacks [269].

By preserving the quadratic character of messages, RSA encryption always leaks at least one bit of information about the plaintext. Lipton [223] shows how this weakness can be used to cheat at a protocol suggested by Shamir, Rivest, and Adleman for playing "mental poker" [292]. A cheater can mark the aces as quadratic residues.

The common modulus RSA cryptosystem is a variation of the RSA cryptosystem in which the same modulus n is used for every key. Only the encryption exponent e varies.[5] In the common modulus RSA cryptosystem, the set of encryption functions form a group under functional composition. As a result, this cryptosystem is vulnerable to the known-plaintext attacks against group ciphers described in chapter 5.

Moreover, the RSA cryptosystem is also vulnerable to these known-plaintext attacks, since the cryptanalyst can proceed as if he were attacking the common modulus RSA. Although the RSA cryptosystem is vulnerable to the meet-in-the-middle and cycling attacks, these attacks are less efficient at breaking the RSA than are known factoring techniques. This fact is evidence that, provided the key space is large enough to withstand an $O(\sqrt{K})$ time attack, group ciphers are not necessarily insecure.

The rich algebraic structure of the RSA cryptosystem makes it possible to perform public-key cryptography and digital signatures. Yet the algebraic structure also allows short-cut solutions over exhaustive search of the key space through permitting an attack against group ciphers and through permitting attacks based on factoring. In the sense that the RSA cryptosystem allows short-cut solutions over exhaustive search of the key space, the RSA cryptosystem does not achieve the same level of security that one would ideally like from a cryptosystem with the same number of keys. At least in the case of the RSA cryptosystem, the special capabilities of public-key cryptography and digital signatures come at the cost of less efficient use of key bits.

[5] As explained in [226], from any RSA key (e, d, n), it is easy to compute the factors of n. For the RSA cryptosystem, this fact causes little trouble since every user has a different modulus. But for the common modulus RSA cryptosystem, this fact is a very serious drawback since it implies that every key holder can factor n.

Chapter 3

Preliminaries

This chapter presents background material helpful in understanding the rest of part I. Section 3.1 introduces the notion of a finite, deterministic cryptosystem and reviews some algebraic properties of this type of cryptosystem. This section also explains some concepts and terminology from permutation group theory used throughout part I. Section 3.2 introduces the Data Encryption Standard (DES), which is a typical example of a finite, deterministic cryptosystem. This section also reviews previous work on DES relevant to the closure tests presented in chapter 4. Finally, Section 3.3 explains the so-called "Birthday Paradox," which plays a crucial role in the closure tests.

3.1 Finite, Deterministic Cryptosystems

A *finite, deterministic cryptosystem* is a cryptosystem with a finite number of messages and keys and whose encryption transformations are deterministic. This class includes many known cryptosystems, including simple substitution, transposition, the Enigma and Hagelin cryptographs [71], DES, and the RSA cryptosystem when used as a conventional cryptosystem. However, this class does not include stream ciphers, probabilistic schemes [138] nor some esoteric systems considered by Brassard [132,153]. We shall now review some basic concepts, definitions, and properties of finite, deterministic cryptosystems.

3.1.1 Definitions

A *(finite, deterministic) cryptosystem* is an ordered 4-tuple (K, M, C, T), where K, M, and C are finite sets called the *key space*, *message space*, and *ciphertext space*, and $T : K \times M \rightarrow C$ is a transformation such that, for each $k \in K$, the mapping $T_k = T(k, \cdot)$ is invertible.

The *order* of a cryptosystem is the number of distinct transformations; the *degree* of a cryptosystem is the size of the message space. A cryptosystem is *endomorphic* if and only if the message space and ciphertext space are the same set.

Thus, for any cryptosystem $(K, \mathcal{M}, \mathcal{C}, T)$, each key $k \in K$ represents a transformation $T_k : \mathcal{M} \rightarrow \mathcal{C}$. In an endomorphic cryptosystem, each key represents a permutation on $\mathcal{M}$. A cryptosystem is *faithful* if and only if every key represents a distinct transformation.

For any cryptosystem $\Pi = (K, \mathcal{M}, \mathcal{C}, T)$, let $\mathcal{T}_\Pi = \bigcup\{T_k : k \in K\}$ be the set of all encryption transformations, and let $G_\Pi = \langle \mathcal{T}_\Pi \rangle$ be the group generated by $\mathcal{T}_\Pi$ under functional composition. For any transformation $T_k \in \mathcal{T}_\Pi$, let T_k^{-1} denote the inverse of T_k. In addition, let $K = |K|$ be the size of the key space; let $M = |\mathcal{M}|$ be the degree of Π; and let $m = |\mathcal{T}_\Pi|$ be the order of Π. Whenever the meaning is clear, we will omit the subscript Π.

Let $\Pi = (K, \mathcal{M}, \mathcal{C}, T)$ be any finite deterministic cryptosystem. Π is *closed* if and only if its set of encryption transformations is closed under functional composition, *i.e.* if and only if for all keys $i, j \in K$ there exists a key $k \in K$ such that $T_i T_j = T_k$.[1] Since every finite cancellation semigroup is a group, Π is closed if and only if $\mathcal{T}_\Pi$ forms a group under functional composition.

Shannon's notion of a pure cipher generalizes the idea of closure to non-endomorphic cryptosystems [150]. Π is *pure* if and only if, for every keys $i, j, k \in K$, there exists a key $l \in K$ such that $T_i T_j^{-1} T_k = T_l$.[2]

Thus, Π is pure if and only if *for every* $T_0 \in \mathcal{T}_\Pi$ the set $T_0^{-1}\mathcal{T}_\Pi$ is closed. Moreover, $T_0^{-1}\mathcal{T}_\Pi$ is closed *for every* $T_0 \in \mathcal{T}_\Pi$ if and only if $T_0^{-1}\mathcal{T}_\Pi$ is closed *for some* $T_0 \in \mathcal{T}$. Every closed cryptosystem is pure, but not every endomorphic pure cryptosystem is closed.[3]

3.1.2 Notation and Terminology

The following standard terminology involving permutation groups and strings is used throughout part I.

Terminology from Permutation Group Theory

Let $\mathcal{M}$ be any finite set (of messages), and let $M = |\mathcal{M}|$ be the cardinality of $\mathcal{M}$. For any permutations g, h on M we denote the composition of g and h by $gh = g[h(\cdot)]$.

[1] Note that we are using the term *closed cipher* to refer to what Shannon called an *idempotent cipher* [150]. Shannon defined a closed cipher to be any cryptosystem with the property that each cryptographic transformation is surjective.

[2] Shannon also required each transformation of a pure cipher to be equally likely.

[3] The restriction of simple substitution [65] on the standard alphabet where the letter 'A' is always mapped to 'B' is an endomorphic system that is pure but not closed.

For any permutations $g_1, g_2, \ldots, g_n$, let $\langle g_1, g_2, \ldots, g_n \rangle$ denote the group generated by $g_1, g_2, \ldots, g_n$ under functional composition. Let I be the identity permutation on M.

There are $M!$ distinct permutations on M. Under functional composition, these permutations form a group known as the *symmetric group on* M.[4] Every permutation can be expressed as a product of transpositions. The *even permutations* are the permutations that can be expressed as an even number of transpositions. There are exactly $M!/2$ even permutations on M, and, under functional composition, these permutations form the *alternating group on* M. Let $\mathcal{A}_M$ and S_M be, respectively, the *alternating group* and *symmetric group* on M.

Let G be any subgroup of S_M, and let x be any message in M.

The *order of* G is the number of elements in G; the *degree of* G is the cardinality of M. For any $g \in S_M$, the *order of* g is the order of $\langle g \rangle$.

The *G-orbit of* x is the set $G\text{-orbit}(x) = \{g(x) : g \in G\}$. For any permutation $g \in S_M$, we will write $g\text{-orbit}(x)$ to denote the $\langle g \rangle$-orbit of x. If f is any function (not necessarily a permutation) and if $x \in \text{Domain}(f)$, we define the *f-closure of* x to be the set $f\text{-closure}(x) = \{f^i(x) : i \geq 0\}$.

The *G-stabilizer of* x is the set $H_x = \{g \in G : g(x) = x\}$, which forms a subgroup of G.

For any subset of permutations $S \subseteq S_M$ and for subset of messages $X \subseteq M$, we say *S acts transitively on X* if and only if, for every pair of messages $x, y \in X$, there exists some transformation $g \in S$ such that $g(x) = y$.

String Terminology

Let Σ be any finite set of symbols. For any integer n, Σ^n denotes the set of strings over Σ of length exactly n. Similarly, Σ^* denotes the set of finite strings over Σ of length 0 or more, and Σ^+ denotes the set of finite strings over Σ of length 1 or more.

For any string $s \in \{0,1\}^*$, let $|s|$ denote the length of s.

For any any string $s \in \{0,1\}^+$, let $\overline{s}$ denote the bitwise complement of s.

For any two strings $s_1, s_2 \in \{0,1\}^+$ of the same length, let $s_1 \oplus s_2$ denote the bitwise exclusive-or of s_1 and s_2.

3.1.3 Algebraic Properties of Closed and Random Ciphers

In this section, we review several important differences between closed cryptosystems and cryptosystems that consist of randomly chosen permutations.[5] These differences will form the basis of the statistical closure tests presented in chapter 4.[6]

[4] See [2], [8], or [9] for a review of basic concepts in permutation group theory.

[5] By "a set of randomly chosen permutations on M," we mean a set of permutations each member of which is chosen independently, with uniform probability from S_M.

[6] This section draws heavily from basic results in permutation group theory and from Shannon's classic paper [150,56].

Since every finite cancellation semigroup is a group [8], any endomorphic cryptosystem is closed if and only if its set of encryption transformations forms a group under functional composition. Thus, closed ciphers have a great deal of algebraic structure. By contrast, one expects a set of randomly chosen permutations to have virtually no algebraic structure, as the following lemmas makes precise.

Properties of cryptosystems can be studied both by examining abstractly the set of encryption transformations and by examining how the transformations act on the message space. Lemma 3.1 captures one important difference between closed and random ciphers by focusing on a property of the set of encryption transformations. This lemma says that if a cryptosystem is closed, then for every transformation T_k there are many pairs T_i, T_j such that $T_k = T_iT_j$; but, if a cryptosystem consists of randomly chosen permutations, then for every transformation T_k it is unlikely to find any pair T_i, T_j such that $T_k = T_iT_j$. This lemma provides the basis of the meet-in-the-middle closure test.

Lemma 3.1 *Let $\Pi = (\mathcal{K}, \mathcal{M}, \mathcal{M}, \mathcal{T})$ be any endomorphic cryptosystem of order m, and let $k \in \mathcal{K}$ be any key. If Π is closed, then there are exactly m pairs of keys $T_i, T_j \in \mathcal{T}_\Pi$ such that $T_iT_j = T_k$. If $\mathcal{T}$ is selected at random from $S_\mathcal{M}$, then the expected number of pairs of transformations $T_i, T_j \in \mathcal{T}_\Pi$ such that $T_iT_j = T_k$ is $m^2/M!$.*

Proof. Part 1: Assume Π is closed. For every transformation $T_i \in \mathcal{T}_\Pi$, there is exactly one transformation $T_j \in \mathcal{T}_\Pi$ such that $T_iT_j = T_k$. Part 2: Assume $\mathcal{T}_\Pi$ is chosen at random. There are m^2 pairs $T_i, T_j \in \mathcal{T}_\Pi$ and each pair has a $1/|S_\mathcal{M}|$ chance of corresponding to T_k. Moreover, these probabilities are independent. ∎

For unfaithful cryptosystems, it is important to distinguish between drawing a transformation from the set of transformations and picking a representation of a transformation from the keyspace. Mathematically, it is usually more convenient to think about selecting a transformation from a set of transformations, but in practice, one must often select a transformation by choosing a key. Let $\mathcal{T}$ be the set of cryptographic transformations in any cryptosystem with keyspace $\mathcal{K}$. If T_k is selected from $\mathcal{T}$ at random, then the probability of picking any particular transformation in $\mathcal{T}$ is exactly $1/m$, where $m = |\mathcal{T}|$. However, if a key k is selected at random from $\mathcal{K}$, then the probability that k represents any particular transformation in $\mathcal{T}$ is between $1/m$ and $1/K$, where $K = |\mathcal{K}|$. If the cryptosystem is unfaithful, then $m < K$.

The next lemma describes the structure imposed on the message space by any closed cipher; specifically, lemma 3.2 says that the orbits of any closed cipher partition the message space into transitive sets. This lemma provides the basis of the cycling closure test.

Lemma 3.2 *Let $\Pi = (\mathcal{K}, \mathcal{M}, \mathcal{M}, \mathcal{T})$ be any endomorphic cryptosystem of order m. If Π is closed, then, for some $1 \le r \le m$, the $\mathcal{T}_\Pi$-orbits of messages in $\mathcal{M}$ partition $\mathcal{M}$ into r mutually disjoint sets $\mathcal{M} = B_1 \cup \cdots \cup B_r$ such that, for each $1 \le i \le r$, the*

following two statements hold:

1. $\mathcal{T}_\Pi$ *acts transitively on* B_i.
2. $|B_i|$ *divides* m; *in fact, for any* $x \in B_i$, $|B_i| = m/|H_x|$, *where* H_x *is the* $\mathcal{T}_\Pi$*-stabilizer of* x.

Proof. (Sketch) For each $x \in M$, consider the left cosets of H_x in $\mathcal{T}$ [8]. ∎

Corollary 3.3 *If DES is closed, then DES partitions its message space into at least* 2^8 *mutually disjoint transitive sets, each of size at most* 2^{56}.
Proof. DES has degree 2^{64}, but order at most 2^{56}. ∎

The next lemma calculates the expected number of spurious decipherments of closed and random ciphers; this lemma is useful in the analysis of the tests.

Lemma 3.4 *Let* $\Pi = (K, M, M, T)$ *be any endomorphic cryptosystem of order* m, *let* $p \in M$ *be any message, let* $k \in K$ *be any key, and let* $c = T_k(p)$. *If* Π *is closed, then the number of transformations that map* p *to* c *is* $m/|B_p| = |H_p|$, *where* B_p *is the* $\mathcal{T}_\Pi$*-orbit of* p, *and* H_p *is the* $\mathcal{T}_\Pi$*-stabilizer of* p. *If* $\mathcal{T}_\Pi$ *is chosen at random, then the expected number of transformations that map* p *to* c *is* m/M.
Proof. Part 1: (Sketch) By lemma 3.2 and the fact that, for any $x, y \in B_p$, $|\{T_i \in \mathcal{T}_\Pi : T_i(x) = y\}| = |\{T_i \in \mathcal{T}_\Pi : T_i(p) = c\}|$. Note that $|H_p| = |H_c|$. Part 2: Each transformation in $\mathcal{T}_\Pi$ other than T_k maps p to c with probability $1/M$. ∎

3.2 The Data Encryption Standard

The Data Encryption Standard (DES) defines a particular endomorphic cryptosystem with $M = C = \{0,1\}^{64}$ and $K = \{0,1\}^{56}$. This section reviews some of the known properties of DES, concentrating on properties relevant to the question "Is DES a group?" and on previous cycling studies.

3.2.1 Background

On November 23, 1976, the United States National Bureau of Standards (NBS) adopted DES as a federal standard for the cryptographic protection of computer data [59,157]. Designed by IBM, DES was based in part on IBM's Lucifer cipher [59]. Both DES and Lucifer consist of a cascade of rounds, each performing a transposition and a nonlinear substitution. NBS selected DES from several competing designs, submitted in response to a 1974 call for proposals for cryptographic algorithms.

Figure 3.1 summarizes the structure of DES, as explained in its official definition.[7] The main part of DES consists of a cascade of 16 rounds, each under the control of a separate 48 bit *round key*, computed from the 56 bit key. The initial and final

[7] Figures 3.1 and 3.2 are taken from [157].

permutations have no cryptographic significance; according to IBM's Carl Meyer, they are simply artifacts from the bonding pattern of the Lucifer chip. Cryptographic strength is built up through the iteration of many rounds. This round structure also supports fast and simple pipelined implementations of the algorithm.[8]

The security of DES rests crucially on its eight "S-boxes," which are the only nonlinear components of the algorithm (see figure 3.2). Although the S-boxes were thoroughly examined by IBM and NSA, the criteria used to select and validate the S-boxes remain unpublished in the open literature. Several unusual structures have been found in the S-boxes [180,165,196,162], but the full implications of these structures remain a mystery.

Controversy surrounds DES, its security, and the manner in which it was designed and adopted. At the center of this controversy is the National Security Agency (NSA), established in 1952 by a classified presidential directive to gather communications intelligence and to safeguard American communications [72,75]. After IBM submitted its proposal, at the request of NBS, NSA evaluated the suitability of using the IBM algorithm as a national standard. NSA participated in the final design of DES, influencing the choice of the crucial "S-box" substitution tables and convincing IBM to shorten the key length from 128 bits to 56 bits [199].

IBM concurred with NSA that the 56 bit key was adequate for the intended applications of DES. But Diffie and Hellman argued that the 56 bit key was too short. They described a $20 million LSI machine that could break DES by exhaustively searching the keyspace [173]. According to Diffie and Hellman [173], an IBM study—subsequently disavowed by IBM—concluded that IBM could deliver such a machine by 1981 at a selling price of $200 million. Although some researchers disagreed with some of Diffie and and Hellman's detailed analysis [190,198], the Diffie and Hellman study showed convincingly that the 56 bit key gave DES a built-in obsolescence. With the rapid advance of computer technology, DES could not remain adequate for its intended applications very long.

In 1987, DES will come up for review by NBS. Already, NSA has informed NBS that NSA will no longer support DES, and NBS has announced that after 1987, it will no longer certify new DES products [82]. Instead, NSA will make available implementations of classified algorithms. The implementations will come in tamper-resistant boxes or chips, and the algorithms will not be revealed. These devices will be developed by industry as part of a special NSA program. Instead of relying on a single encryption algorithm, NSA will make available several different algorithms. NSA will also provide secret keys for those who would like to use them. This transition will likely cause much turmoil in the banking community.

Because DES has degree 2^{64}, but order at most 2^{56}, DES is intransitive. It is

[8] Like DES, the RSA cryptosystem also builds up cryptographic strength through a cascade of primitive cryptographic operations. With RSA, each primitive operation is modular multiplication.

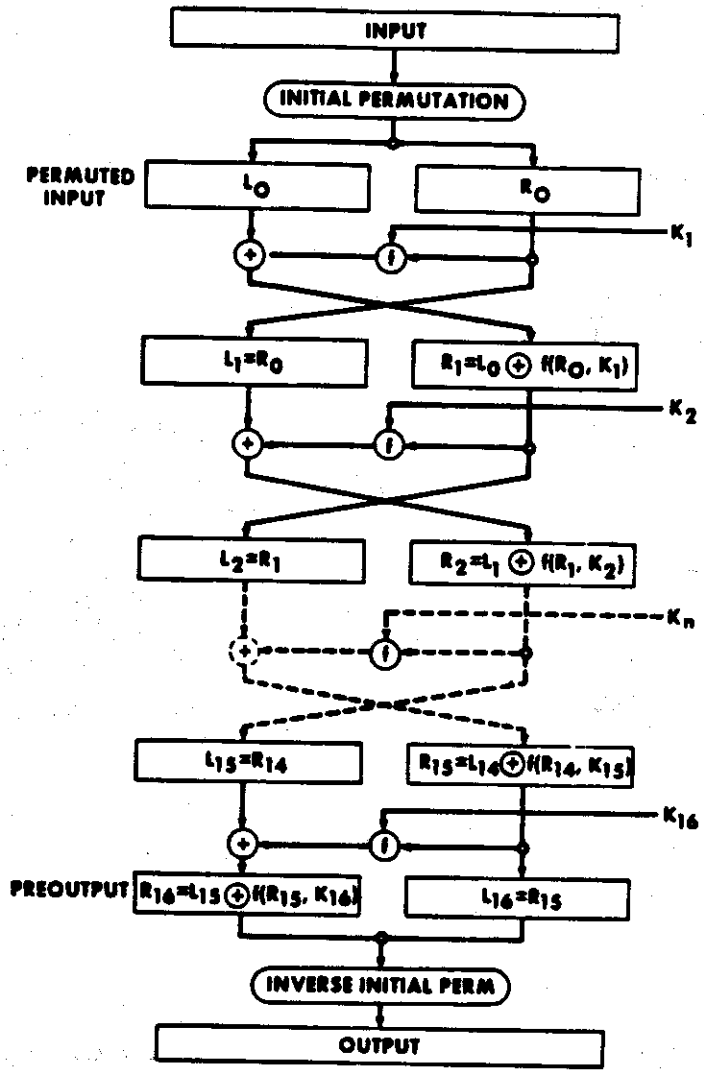


Figure 3.1: DES is a cascade of 16 rounds, each under the control of a separate 48 bit round key derived from the 56 bit key.

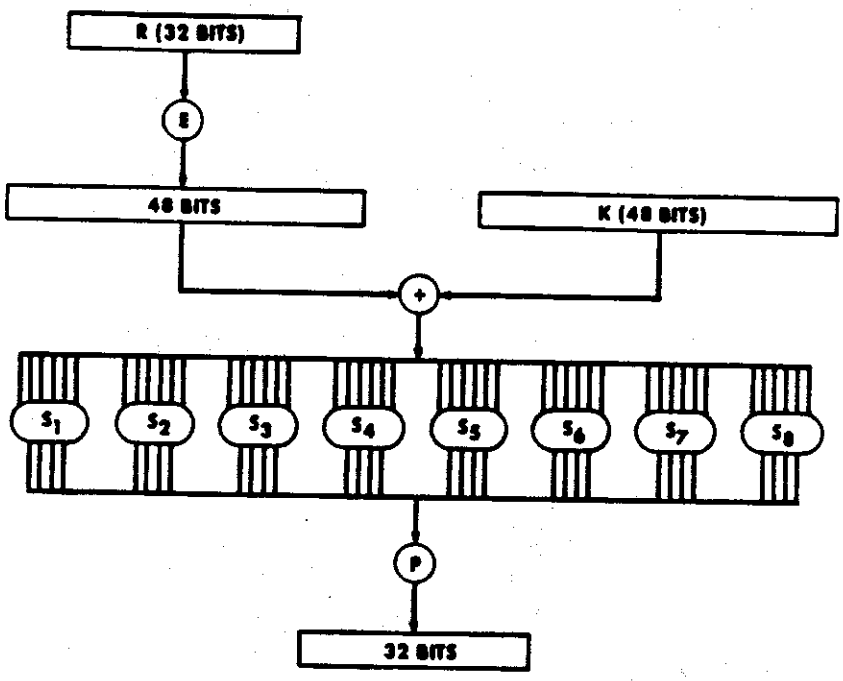


Figure 3.2: Computation of DES's nonlinear f function

unknown if DES is faithful, closed, or pure. It is also unknown whether or not any DES transformation is the identity permutation.

3.2.2 Is DES a Group?—A Priori Beliefs

The question of whether or not DES is closed is a question about the order of the group generated by DES. Grossman and Coppersmith observed that $G_{DES} \subseteq A_M$ [164], but no one has disproved the possibility that $G_{DES} = T_{DES}$.[9]

There are several reasons to suspect DES is not closed. First, Coppersmith and Grossman proved "DES-like" permutations generate the alternating group [164].[10] Second, if even just two permutations are chosen at random from S_M, then there is an overwhelming chance (greater than $1-e^{-\sqrt{M}}$) that these permutations generate either A_M or S_M [86,89]. Third, no one has announced finding any three keys $i, j, k \in K$ such that $T_k = T_i T_j$. Finally, according to a 1977 unclassified summary of a report of the Senate Select Committee on Intelligence, the National Security Agency certified that "the final DES algorithm was, to the best of their knowledge, free of any statistical or mathematical weaknesses" [199].

On the other hand, DES is not a set of randomly chosen permutations, and Coppersmith and Grossman did not prove that DES generates A_M. Furthermore, DES is known to have the following three regularities [59,68,165,180].

1. *Complementation Property.* For every key k and every message x, $T_{\overline{k}}(\overline{x}) = \overline{T_k(x)}$.

2. *Existence of Weak Keys.* There exist at least four distinct keys k such that $T_k^2 = I$.

3. *Existence of Semi-Weak Keys.* There exist at least six distinct pairs of keys $k_1 \neq k_2$ such that $T_{k_2} T_{k_1} = I$.

The last two properties, however, apparently involve only a small fraction of the total number of DES transformations. While many people may have a strong belief that DES is not closed, there is a need for convincing objective evidence to answer this question.

3.2.3 Previous Cycling Studies on DES

To the best of our knowledge, only three other cycling experiments on DES have been reported in the open literature. These experiments were performed by Gait; Davies and Parkin; and Hellman and Reyneri. Each of these experiments differs from

[9] To see that $G_{DES} \subseteq A_M$, note that each round of DES is an even permutation.

[10] See [176] for an extension of this result.

our cycling closure test, and none of these previous experiments determined if DES generates a small group.

The analysis of each of these previous experiments depends heavily on the following two facts [92,93] ([27], exercise 3.1.12]). Let $x_0 \in M$ be any message. For a randomly selected *function* f on M, the expected size of f-closure(x_0) is about $\sqrt{M}$. (This follows from the Birthday Paradox.) But for a randomly selected *permutation* g on M, the expected size of g-orbit(x_0) is about $M/2$. (This is true because, for any $1 \leq l \leq M$, the probability that the cycle containing x_0 has length exactly l is $1/M$, independent of l.)

Gait [175] investigated the statistical properties of pseudo-random key streams produced by DES in output-feedback mode [158]. Provided the feedback width is exactly 64 bits, each such key stream describes the orbit of a DES transformation on some initial message. In a series of software experiments, Gait computed the key stream produced by DES in output-feedback mode to at most $10^6 \approx 2^{20}$ places. Gait found no cycles for nonweak keys.[11] Unfortunately, Gait did not state what feedback width he used. Gait also proposed a new power-spectrum test for nonrandomness and applied it to each of the pseudo-sequences he computed from non-weak keys. Gait observed that each of these sequences was considered random by his test.

Provided a feedback width of 64 bits is used, the cycling study considered by Gait can be viewed as a closure test. If DES were closed, then each of the orbits considered by Gait would have at most $K = 2^{56}$ messages (see lemma 3.2). Hence, observing an orbit of length greater than 2^{56} would be direct proof that DES is not closed. Although we will not do so in this thesis, it is also possible to interpret Gait's orbit test as a statistical closure test. Viewed as a statistical closure test, the orbit test can be strengthened by combining the test with tests for other algebraic properties (see section 7.2).

Davies and Parkin [167] [166] and Jueneman [184] studied mathematically the cycle structure of the key stream produced in output-feedback mode. Each of these studies concluded that, if DES is used in output-feedback mode with a feedback-width of less than 64 bits, then the resulting key stream will cycle in about 2^{32} steps, on the average (the exact expected cycle length depends slightly on the feedback width). If all 64 bits are fed back, then the expected cycle length is about 2^{63}. The point is that the state transition function in output-feedback mode is a permutation if and only if all 64 bits are fed back. Although Davies and Parkin did not report performing any experiments on the full DES algorithm, Davies and Parkin did run a series of experiments on DES substitutes consisting of random permutations on $\{0,1\}^8$. Their experimental results agreed with their theoretical predictions.

In an attempt to better understand how effectively the Hellman cryptanalytic time-space tradeoff [261] could be applied to DES, Hellman and Reyneri [183] exam-

[11] Since $T_k^2 = I$ for any weak key k, the key stream produced in output-feedback mode with feedback width 64 bits cycles after 128 bits whenever a weak key is used.

ined the cycle structure of mappings induced by DES on the keyspace. Specifically, they considered mappings $F_x : \mathcal{K} \to \mathcal{K}$ defined by $F_x(k) = \rho(T_k(x))$, where $\rho : \mathcal{M} \to \mathcal{K}$ is a projection[12] and $x \in \mathcal{M}$ is some fixed message. Their studies detected no significant statistical irregularities. Whether or not DES is closed, the expected cycle length of the Hellman/Reyneri experiment is about $\sqrt{K} = 2^{28}$.

Each of these previous cycling projects studied the behavior of the powers of some indexed function (*i.e.* $T_k^i(x_0)$ or $F_x^i(k_0)$ for $i = 1, 2, \ldots$) where the index of the function was held fixed throughout the experiment: Gait and Davies and Parkin held the key fixed; Hellman and Reyneri held the message fixed. By contrast, our cycling test computes the sequence $x_i = T_{k_i} T_{k_{i-1}} \ldots T_{k_1}(x_0)$ for $i = 1, 2, \ldots$ where at each step i the key k_i is chosen as a pseudo-random function of the previous ciphertext x_{i-1}.

3.3 The Birthday Paradox

This section briefly reviews the "Birthday Paradox" [11], which plays a dominant role in the analysis of the algebraic tests in chapter 4.

The Birthday Paradox involves the question, "If r people are selected at random, what is the chance that no two people will have the same birthday?" Let p_r be this chance. If birthdays are independently and uniformly distributed between 1 and m, then $p_r \approx 1 - \frac{1}{m}\binom{r}{2}$, since there are $\binom{r}{2}$ pairs of people and each pair has a $1/m$ chance of having the same birthday. This approximate analysis, however, ignores the possibility that more than two people might have the same birthday. The "paradox" is that many students are surprised to learn that the probability p_r is so low: with only $r = \sqrt{m}$ people, the odds are approximately .5 that at least two people will have the same birthday.

More exactly,

$$p_r = \frac{(m)_r}{m^r} = \frac{m!}{m^r(m-r)!} \approx e^{-r^2/(2m)} \tag{3.1}$$

where $(m)_r = m(m-1)\cdots(m-r+1)$. The approximation in equation 3.1, and other similar approximations, can be obtained from Stirling's formula [11,34]. For any constant $c > 0$, if $r = c\sqrt{m}$ and m is sufficiently large,

$$p_r \approx e^{-c^2/2}. \tag{3.2}$$

Thus, by choosing $r = c\sqrt{m}$ with c sufficiently large, p_r can be made as small as desired.

The meet-in-the-middle test uses a variation of the Birthday Paradox in which two samples X and Y, each of size r, are drawn at random from a universe of m

[12]Hellman and Reyneri used the projection that removes each of the 8 parity bits.

elements. If X and Y each are drawn without replacement, and if each element is drawn independently with probability $1/m$, then the chance that X and Y do not intersect is exactly $(m)_{2r}/\left((m)_r\right)^2$. If $r = c\sqrt{m}$, this chance is approximately e^{-3c^2}.

Chapter 4

Testing Cryptosystems for Algebraic Structure

This chapter presents two statistical tests for determining if a cryptosystem is closed under functional composition. The first test is a meet-in-the-middle algorithm that uses $O(\sqrt{K})$ time and space. The second test is a novel cycling algorithm that uses $O(\sqrt{K})$ time, but only a small constant amount of space. Each test is based heavily on the Birthday Paradox (see section 3.3). The chapter concludes with brief descriptions of several other related tests. Although our primary interest in these tests was to test DES for closure, purity, and other extreme algebraic weaknesses, the tests are general in nature.

4.1 Conducting and Interpreting the Algebraic Tests

This section describes the nature of our tests, concentrating on the framework in which the tests operate and on how to interpret the test results.

4.1.1 Testing Framework

Input to each test is a finite, deterministic cryptosystem $\Pi = (K, M, C, T)$, with the encryption transformation T presented as a "black box." Given any key $k \in K$ and any message $x \in M$, the box computes $T_k(x)$ and $T_k^{-1}(x)$. No additional information about T is provided. To ensure that messages and keys are easy to detect, generate, and compare, we assume that $M = \{0,1\}^u$, $K = \{0,1\}^v$, and $C = \{0,1\}^w$, for some u, v, w also provided to the test.

We assume that the sets K, M, and C are so large that they cannot be exhaustively searched; each test must proceed by examining a limited number of messages and

keys.

4.1.2 Interpreting the Results

Each closure test computes a statistic, which can be used to calculate a measure of our relative degree of belief in the following two competing hypotheses:

- H_G = "$\mathcal{T}_\Pi$ is a group."
- H_R = "Each transformation T_k was chosen independently with uniform probability from the symmetric group on $\mathcal{M}$."

To compute this measure, we will apply the *theory of the weight of evidence*, as explained by Good [14,12].

Let E be experimental evidence produced by one trial of one of the closure tests. From this evidence we can compute the conditional probabilities $P(E \mid H_G)$ and $P(E \mid H_R)$, as explained in the next two sections. Note, however, that neither closure test enables us to compute $P(E \mid \overline{H_G})$ or $P(E \mid \overline{H_R})$, where $\overline{H_G}$ and $\overline{H_R}$ are the complements of H_G and H_R respectively. Thus, on the basis of experimental evidence, we would be able to conclude only that Π is *not* closed or that Π has a structure different from that expected from a set of randomly chosen permutations; we would not be able to conclude that Π is closed. In the worst case, Π could be closed, except for some isolated pair of keys a, b such that $T_b T_a$ is not in $\mathcal{T}$, even though there exists some key k and some message x_0 such that $T_b T_a(x) = T_k(x)$ for all messages $x \in \mathcal{M}$, $x \neq x_0$.

Initially, each person may have some (subjective) degrees of belief $P(H_G)$ and $P(H_R)$ in hypotheses H_G and H_R respectively. From these initial degrees of belief, each person can compute $O(H_G/H_R) = P(H_G)/P(H_R)$ as his or her initial *odds in favor of H_G over H_R*. After seeing any experimental evidence E, however, each rational person should update his or her own odds in favor of H_G over H_R.

Given any evidence E, a Bayesian would update his or her odds in favor of H_G over H_R as follows:

$$O(H_G/H_R \mid E) \longleftarrow \frac{P(E \mid H_G)}{P(E \mid H_R)} O(H_G/H_R). \tag{4.1}$$

where $O(H_G/H_R \mid E)$ is the *odds in favor of H_G as opposed to H_R given E.*

We encourage the reader to update his or her own odds in favor of H_G over H_R in light of the evidence presented in chapter 6.

4.2 Meet-in-the-Middle Closure Test

The meet-in-the-middle closure test (MCT) works as follows: given any endomorphic cryptosystem $\Pi = (K, M, M, T)$, pick any key $k \in K$ and search for keys $a, b \in K$ such that $T_k = T_b T_a$. If Π is closed, then such a pair of keys a, b can be efficiently found, with high probability. If $\mathcal{T}_\Pi$ is selected at random, then it is unlikely to find any such pair.

To search for a pair of keys $a, b \in K$ such that $T_k = T_b T_a$, we use a standard "meet-in-the-middle" attack similar to that described in [307], for example. Specifically, choose $2r$ keys $a_1, a_2, \ldots, a_r$ and $b_1, b_2, \ldots, b_r$ at random and look for a pair of keys a_i, b_j for some $1 \leq i, j \leq r$ such that $T_k = T_{b_j} T_{a_i}$. To find such a match, represent the cryptographic transformations by their images or preimages of some particular message. Specifically, pick any message $p \in M$, calculate $c = T_k(p)$, and compute $x_i = T_{a_i}(p)$ and $y_i = T_{b_i}^{-1}(c)$, for $1 \leq i \leq r$. Then, look for matches $x_i = y_j$ by sorting the triples $(x_i, a_i, \text{"A"})$ and $(y_j, b_j, \text{"B"})$ for $1 \leq i, j \leq r$ on their first components. Screen out false matches by testing if $T_k(p_h) = T_{b_j} T_{a_i}(p_h)$, for all $1 \leq h \leq l$, for a small number of additional messages $p_1, p_2, \ldots, p_l \in M$. (A false match is a pair of keys $a', b' \in K$ such that $T_k(p) = T_{b'} T_{a'}(p)$ even though $T_k \neq T_{b'} T_{a'}$.) Figure 4.1 summarizes this process

Proposition 4.1 summarizes the main properties of MCT. Informally, proposition 4.1 says that MCT is likely to find a match if Π is closed, but MCT is unlikely to find a match if Π is chosen at random. These facts follow from lemma 3.1 and the Birthday Paradox.

Proposition 4.1 *If Π is closed, then MCT finds a match with probability at least $1 - e^{-3r^2/K}$. If $\mathcal{T}_\Pi$ is chosen at random, then we expect MCT to find a match with probability at most $K^2/M!$.*

Proof. If Π is closed, then for each $1 \leq j \leq r$, $T_{b_j}^{-1} T_k \in \mathcal{T}_\Pi$. In this case, the situation is a variation of the Birthday paradox in which we are drawing two samples X and Y, each of size r, from an urn containing m elements, where m is the order of Π. The first sample consists of the transformations $T_{a_1}, \ldots, T_{a_r}$; the second consists of the transformations $T_{b_1}^{-1} T_k, \ldots, T_{b_r}^{-1} T_k$. If Π is faithful, each element is drawn with probability exactly $1/K$; otherwise, each element is drawn with probability at least $1/K$. Thus, the worst case is when Π is faithful. We are interested in the probability that the samples overlap.

If $\mathcal{T}_\Pi$ is chosen at random, then by lemma 3.1, for any $T_k \in \mathcal{T}$, we expect $\mathcal{T}_\Pi$ to contain a pair $T_a, T_b \in \mathcal{T}$ such that $T_k = T_b T_a$ with probability at most $K^2/M!$. ∎

Thus, by choosing $r = c\sqrt{m}$ with c sufficiently large, we can make the probability $q_r \approx 1 - e^{-3c^2}$ of finding a match if Π is closed as large as desired.

The analysis in proposition 4.1 assumes that each sequence of keys $a_1, \ldots, a_r$ and $b_1, \ldots, b_r$ was drawn without replacement. If these sequences are drawn with

input: An endomorphic cryptosystem $\Pi = (K, M, M, T)$ and
integer control parameters r, l.

begin

1. Pick $k \in K$ and $p_1, \ldots, p_l \in M$ at random. For $i = 1$ to l, compute $c_i = T_k(p_i)$. Let $p = p_1$ and $c = c_1$.

3. For $i = 1$ to r, select $a_i, b_i \in K$ at random and compute $x_i = T_{a_i}(p)$ and $y_i = T_{b_i}^{-1}(c)$.

3. Sort the triples $(x_i, a_i, \text{"}A\text{"})$ and $(y_i, b_i, \text{"}B\text{"})$ for $1 \leq i \leq r$ on their first components.

4. For each "match" $x_i = y_j$ with $1 \leq i, j \leq r$, if $T_k = T_{b_j} T_{a_i}$, then **return**("Match found"). To test if $T_k = T_{b_j} T_{a_i}$, statistically verify that $c_h = T_{b_j} T_{a_i}(p_h)$, for all $1 \leq h \leq l$.

5. **return**("No match found")

end

Figure 4.1: Meet-in-the-middle closure test (MCT)

replacement, then the expected number of samples required to obtain r distinct keys is $K \log((K+.5)/(K-r+.5))$. This situation is a variation of the "collector's problem" [11].

To carry out MCT efficiently, it is important that the expected number of false matches be small. As shown by lemma 3.4, if Π is closed, then at most $(K-1)/|B_p|$ keys other than k map p to c, where B_p is the $\mathcal{T}_\Pi$-orbit of p. If $\mathcal{T}_\Pi$ is chosen at random, then we expect at most $(m-1)/M$ keys other than k to map p to c. Thus, provided K is not too much larger than M, the expected number of false matches is small.

MCT requires $O(r)$ steps and $O(r)$ words of memory. The two most time consuming operations are generating and sorting the lists $x_1, x_2, \ldots, x_r$ and $y_1, y_2, \ldots, y_r$. The required number of encryptions is $2r$ plus the number of additional evaluations used to screen out false matches. If sorting is performed in main memory using radix sort, then sorting will take $O(r)$ machine operations; otherwise, $O(r \log r)$ external memory operations would be needed. The main difficulty with carrying out this test on DES is the high space requirement.

Given the high space requirement of MCT, in practice it may be helpful to use variations of this test that involve time-space tradeoffs. For example, the test could be repeated several times with small values of r. Alternately, the test could build a small hash table for the x_i's and then lookup each y_i in the table without saving the y_i's. If encryption is relatively fast in comparison to the other required operations, then it might be advantageous to save only those $x_i's$ that fall into some subset of the message space. Parallel variations of MCT are also possible.

4.3 Cycling Closure Test

Given any endomorphic cryptosystem $\Pi = (K, \mathcal{M}, \mathcal{M}, T)$, the cycling closure test (CCT) takes a pseudo-random walk in $\mathcal{M}^l$ for some small l. The walk continues for a specified number of steps or until a cycle is encountered. Long walks are strong evidence that Π is not closed; short walks are strong evidence that Π has a structure different from that expected from a set of randomly chosen permutations.

Specifically, CCT picks an initial vector of messages $\hat{x}_0 \in \mathcal{M}^l$ at random and computes the leader length and cycle length of a sequence $\hat{x}_0, \hat{x}_1, \ldots$. For each $i \geq 0$, the next element in this sequence is computed by

$$\hat{x}_{i+1} = f_\rho(\hat{x}_i) \tag{4.2}$$

where the function $f_\rho : \mathcal{M}^l \to \mathcal{M}^l$ is defined by

$$f_\rho(\hat{x}) = T_{\rho(\hat{x})}(\hat{x}) \tag{4.3}$$

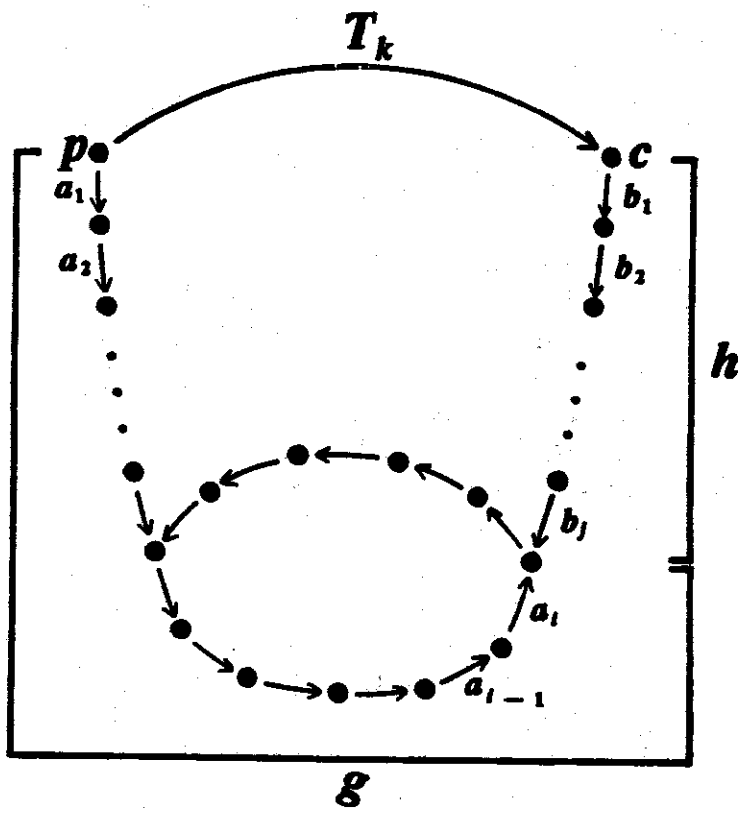


Figure 5.1: Cycling known-plaintext attack

Chapter 6

Experimental Work on DES

Using a combination of software and special-purpose hardware, we applied the cycling closure test and other algebraic tests to DES. Organized in four sections, this chapter describes our experimental work. Section 6.1 summarizes our results. Section 6.2 explains two structural findings. Section 6.3 describes our special-purpose hardware. Section 6.4 gives detailed descriptions of our results. This experimental work was carried out jointly with Burton Kaliski and Ronald Rivest.

6.1 Summary of Experimental Results

On April 4, 1985, we completed our first trial of the cycling closure test. This single experiment gives strong evidence that DES is not closed. During May through August 1985, we performed additional experiments, including two more closure tests, one extended message space closure test, two purity tests, and two orbit tests. Results from Seven of these experiments were consistent with the hypothesis that DES acts like a set of randomly chosen permutations. In particular, these experiments gathered overwhelming evidence that DES is neither closed not pure. But one orbit experiment involving the composition of two weak keys unexpectedly encountered a small cycle, which was the result of hitting fixed points for each of the weak keys.

Table 6.1 summarizes our experimental findings. For each experiment, the table lists the approximate leader length and cycle length encountered. The sums of these lengths form the values of the statistic ω computed by the tests. The table also lists the conditional probabilities p_G, p_R, P_G, and P_R of the experimental evidence under the hypotheses H_G and H_R respectively. The numbers p_G and p_R are based on probability density calculations and indicate the chance of encountering a cycle after exactly r steps, where r is the observed value of ω. The numbers P_G and P_R are based on probability distribution calculations and indicate the chance of not encountering a cycle within r steps.

For experiments 1, 2, 3, 5, and 6, p_G and p_R were computed from equations 4.7

No.	Experiment	|Leader|	|Cycle|	p_G	p_R	P_G	P_R
1	Closure	$\approx 2^{25}$	$\approx 2^{33}$	$\leq 10^{-193}$	$\approx 10^{-10}$	$\leq 10^{-193}$	≈ 0.17
2	Closure	$\approx 2^{30}$	$\approx 2^{33}$	$\leq 10^{-264}$	$\approx 10^{-10}$	$\leq 10^{-264}$	≈ 0.09
3	Closure	$\approx 2^{31}$	$\approx 2^{30.5}$	$\leq 10^{-41}$	$\approx 10^{-10}$	$\leq 10^{-41}$	≈ 0.69
4	Ext. closure	(no cycle in 2^{34} steps)				$\leq 10^{-889}$	$\approx 1 - 10^{-18}$
5	Purity	$\approx 2^{31.5}$	$\approx 2^{30}$	$\leq 10^{-61}$	$\approx 10^{-10}$	$\leq 10^{-61}$	≈ 0.57
6	Purity	$\approx 2^{30}$	$\approx 2^{32}$	$\leq 10^{-94}$	$\approx 10^{-10}$	$\leq 10^{-94}$	≈ 0.43
7	Weak key orbit	0	$\approx 2^{33}$	*	$\approx 10^{-19}$	*	$\approx 10^{-9}$
8	Orbit	(no cycle in 2^{36} steps)				*	$\approx 1 - 10^{-8}$

* Depends on hypothesized group structure.

Table 6.1: Summary of DES experiments, May–August, 1985. The numbers p_G, p_R, P_G and P_R are the conditional probabilities of the experimental evidence under the hypotheses "DES is closed (pure)" and "Each DES transformation was drawn at random from the symmetric group on $\mathcal{M}$" respectively. The numbers p_G and p_R indicate the chance of encountering a cycle after exactly r steps, where r is the sum of the observed leader and cycle lengths. The numbers P_G and P_R indicate the chance of not encountering a cycle within r steps.

and 4.6 respectively. For these same experiments, as well as for experiment 4, P_G and P_R were computed from equations 4.4 and 4.5 respectively. For experiments 7 and 8, the values of p_R and P_R were computed as explained in section 4.4.3. As explained in section 4.3, for simplicity, we coarsely bound p_G from above by P_G.

In the first cycling closure experiment, we found a cycle of length exactly $\mu = 7,985,051,916$ with a leader of length $\lambda = 34,293,589$. This experiment ran for about two and a half days. Let E denote the evidence from our experiment. Since $\mu + \lambda \approx 2^{33} = 2\sqrt{M} = 32\sqrt{K}$, it follows from equation 4.8 that $P(E \mid H_G)/P(E \mid H_R) \leq \left(e^{-32^2/2}/e^{-2^2/2}\right) \cdot (2^{64}/2^{33}) \leq e^{-510+22} = e^{-488}$. On the basis of this experiment alone, each reader should decrease his or her odds in favor of H_G over H_R by a factor of about e^{-488}. Results of the other closure and purity experiments can be interpreted in a similar fashion.

The second closure experiment produced even stronger evidence that DES is not closed. Moreover, the pseudo-random walks from the first two experiments drained into the same cycles. See section 6.2.1.

Using 128-bit messages, the extended closure test did not cycle after 2^{34} steps, showing that the group generated by DES probably has at least 2^{68} elements. This experiment ran for 10 days.

In experiment 7, we computed the orbit of the composition of the two weak keys that consist respectively of all zeros and all ones. This experiment produced a

short cycle of approximately 2^{33} steps, which would be unusual (the probability of encountering a cycle of length at most 2^{33} is less than 10^{-9}) if the tested permutation were chosen at random from $S_{\mathcal{M}}$. See sections 6.2.2.

In experiment 8, we computed the orbit of a randomly chosen transformation for two weeks. No cycle cycle was detected after 2^{36} steps. This experiment provided no evidence of any algebraic weakness.

In addition, we ran one reduced message space test for which we observed no algebraic weaknesses.

As one test of correctness, we ran a software implementation of the cycling closure test for 30,000 steps. The software and hardware implementations agreed on all values. As a second test of correctness, we repeated each experiment and obtained identical results. We invite the interested reader to verify our results using the detailed experimental data found in section 6.4.

6.2 Two Structural Findings

Although most of our experimental results are consistent with the hypothesis that DES acts like a set of randomly chosen permutations, three experiments did yield interesting regularities. One regularity is a result of the well-known complementation property; the other involves a newly discovered property of the weak keys. We will now explain these structural findings.

6.2.1 Complementation and Drainage Properties

In the first two experiments, we performed two independent trials of the cycling closure test. Each of these experiments used the "identity" next key function—the function $\rho: \mathcal{M} \rightarrow \mathcal{K}$ that removes each of the eight parity bits. These two experiments produced two interesting findings. First, each of the pseudo-random walks drained into the same cycle. Second, each point on the cycle was the bitwise complement of the corresponding point exactly halfway around the cycle. Figure 6.1 illustrates these findings.

The first finding is explained by the fact that, for the graph of a randomly chosen function, most points on the graph will probably drain into the same cycle. See [183] for one analysis of this phenomenon.

The second finding is a consequence of DES's complementation property[1] and the fact that the identity next key function also has a complementation property: for all messages x, $\rho(\overline{x}) = \overline{\rho(x)}$. The cycling closure test computes a pseudo-random walk $x_0, x_1, \ldots$, where $x_{i+1} = T_{\rho(x_i)}(x_i)$, for $i \geq 1$. If $x_i = \overline{x_j}$ for any $i > j$, then it would

[1] For every key k and every message x, $T_k(x) = \overline{T_{\overline{k}}(\overline{x})}$ [59].

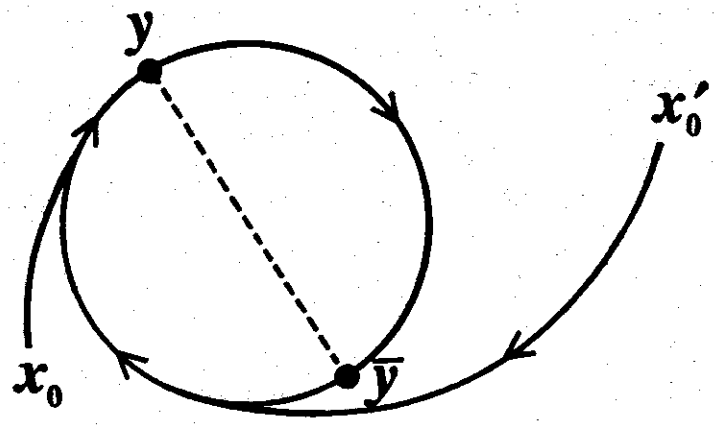


Figure 6.1: Results of experiments 1 and 2. Starting at different initial messages, both pseudo-random walks entered the same cycle. Every message on the cycle is the bitwise complement of the corresponding message halfway around the cycle.

follow that

$$x_{i+1} = T_{\rho(x_i)}(x_i) = T_{\rho(\overline{x_j})}(\overline{x_j}) = T_{\overline{\rho(x_j)}}(\overline{x_j}) = \overline{T_{\rho(x_j)}(x_j)} = \overline{x_{j+1}}\,. \tag{6.1}$$

Therefore, by induction, $x_{i+h} = \overline{x_{j+h}}$ for all $h \geq 0$. This situation arises whenever some $x_i = \overline{x_j}$ before any $x_i = x_j$ with $i > j$, which will happen for about half of all initial messages.

6.2.2 Fixed Points of the Weak Keys

In experiment 7, we computed the orbit of a message under the composition of the two weak keys that consist respectively of all zeros and all ones. Let these two keys be denoted by w_0 and w_1 respectively. Although each weak key transformation is self-inverse, we did not expect the composition $T_{w_1}T_{w_0}$ to produce short orbits. Much to our surprise, we detected a cycle of length less than 2^{33}. We presented this finding at the Crypto 85 conference and sought a simple explanation.

After some thought, Don Coppersmith suggested that we had encountered fixed points of the weak keys, *i.e.*, messages x, y for which $T_{w_0}(x) = x$ and $T_{w_1}(y) = y$. Figure 6.2 illustrates the effect of the fixed points on experiment 7 and explains why a cycle resulted. Experiment 7 computes the ψ-orbit of an initial message x_0, where $\psi = T_{w_1}T_{w_0}$. Let $x_i = \psi^i(x_0)$, for all $i \geq 1$. In figure 6.2, filled circles denote the messages $x_0, x_1, \ldots$ in the ψ-orbit of x_0. Unfilled circles denote intermediate

messages $y_i = T_{w_0}(x_i)$, for all $i \geq 0$. After encountering a fixed point for T_{w_0} on the j-th step ($j \approx 2^{32}$), the walk began to retrace its steps "out of phase" in the sense that $x_{j+i} = y_{j-i}$ for all $i \geq 0$. Continuing in this fashion, the walk passed over the initial message x_0 in a "hidden crossing" $y_{2j} = x_0$, unnoticed during the experiment since the intermediate values y_i were not examined. After approximately 2^{32} steps past the hidden crossing, the walk encountered a fixed point for T_{w_1}. Again, the walk retraced its steps, but this time in phase, finally returning to the initial message x_0.

As we will show, for each weak key, a fixed point results whenever the L and R registers of DES agree after eight rounds.[2] Assuming that the distribution of values taken on by the 32-bit L and R registers is random after eight rounds, the L and R registers will agree after eight rounds with probability $1/2^{32}$. Hence, since there are 2^{64} messages, we expect there to be approximately $2^{64} \cdot 2^{-32} = 2^{32}$ fixed points for each weak key.

To understand why a fixed point results for each weak key whenever the L and R registers agree after eight rounds, it is helpful to describe DES as a product of permutations

$$T_k = P^{-1}\pi(\pi h_{k_{16}}) \cdots (\pi h_{k_1})P, \tag{6.2}$$

where k is the 56-bit key, P is the initial permutation, and $k_1, \ldots, k_{16}$ are the sixteen 48-bit round keys derived from k. If k is weak, then $k_1 = k_2 = \cdots = k_{16}$. For all $1 \leq i \leq 16$, the i-th round consists of the permutation πh_{k_i} where $\pi, h_{k_i} : \mathcal{M} \to \mathcal{M}$. It is especially convenient to define π and h_{k_i} in terms of their effects on the L and R registers. For any $r, s \in \{0,1\}^{32}$, π is the "swap" function

$$\pi(r, s) = (r, s) \tag{6.3}$$

and h_{k_i} is the function

$$h_{k_i}(r, s) = (r \oplus f_{k_i}(s), s), \tag{6.4}$$

where f is DES's nonlinear function defined in figure 3.2. Note that, for all round keys k_i, both π and h_{k_i} are self inverse.

Let x be any message and let k be any weak key. If, during the computation of $T_k(x)$, the L and R registers agree after eight rounds, then the effect of rounds eight through nine on the computation of $T_k(x)$ is

$$(\pi h_{k_0})(\pi h_{k_8}) = (\pi h_{k_0})\pi(\pi h_{k_8}) = \pi h_{k_0} h_{k_8} = \pi h_{k_8} h_{k_8} = \pi. \tag{6.5}$$

By similar argument, it then follows that the effect of rounds seven through ten is also π. By induction, it follows that the effect of rounds one through sixteen is π. Hence

[2] See figure 3.1 for an explanation of the L and R registers.

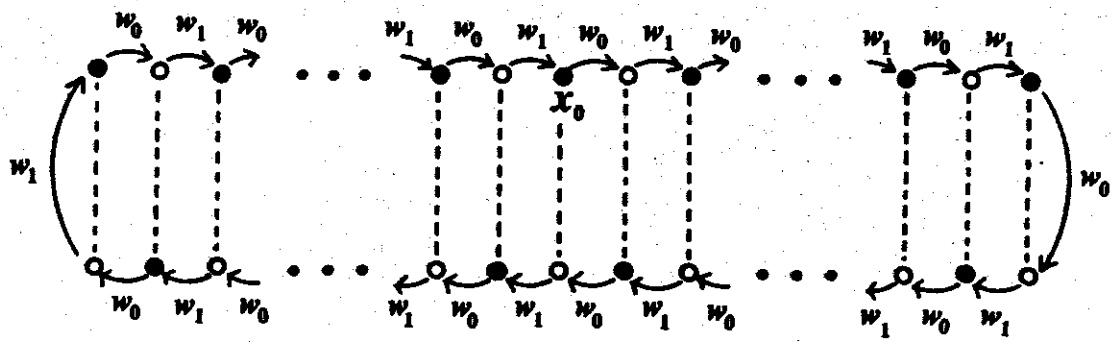


Figure 6.2: Experiment 7 discovered fixed points of the weak keys. Let w_1 and w_0 denote respectively the weak keys that consist of all 1's and all 0's. Filled circles denote the messages x_i on the $T_{w_1}T_{w_0}$-orbit of an initial message x_0. Unfilled circles denote intermediate values $T_{w_0}(x_i)$. Dotted lines link identical messages.

$$T_k(x) = (P^{-1}\pi(\pi)P)(x) = x. \tag{6.6}$$

Note that fixed points arise not only when the round keys are equal, but also when they are "palindromic" in the sense that $k_{8-i} = k_{8+i}$ for all $0 \leq i \leq 8$.

After the conference, we found the fixed points and thus confirmed Coppersmith's hypothesis (see section 6.4). To the best of our knowledge, these fixed points are the first published in the open literature. These fixed points further illustrate the deficiencies of the weak keys.

Coppersmith also suggested that the algebraic structure detected in experiment 7 can be used to prove strong lower bounds on the size of the group generated by DES. Experiment 7 computed the length, l, of the ψ-orbit of x_0, where $\psi = T_{w_1}T_{w_0}$ and x_0 is the initial message. Since l divides the order of ψ, it follows that l divides the order of the group generated by DES. Therefore, if experiment 7 were repeated r times with different initial messages, and if these experiments yielded orbit lengths $l_1, l_2, \ldots, l_r$, then $\text{lcm}(l_1, l_2, \ldots, l_r)$ would be a lower bound on the order of the group generated by DES. We have not yet extended our results in this direction.

Motivated by our findings, Moore and Simmons are carrying out additional experiments to investigate the cycle structure of the weak and semi-weak keys [192].

6.3 Cycling Hardware

We carried out each experiment on an IBM Personal Computer equipped with special-purpose hardware. Our hardware can compute a sequence of 2^{32} DES encryptions per day, where at each step the previous ciphertext is encrypted under a key that depends on the previous ciphertext. This hardware was designed and built by Burton Kaliski, with some help from Leon Roisenberg [187].

Our goal was to implement the cycling closure test in the simplest way that would enable us to carry out each trial of the experiment within a few days. For each experiment, we needed to compute about 2^{32} encryptions, changing the key at each step. For this application, software implementations of DES are too slow.[3] Moreover, commercially available DES boards are not suited for our purposes: to compute and load a new key for each encryption would require interaction by the host computer, introducing tremendous overhead. Therefore, we built our own hardware.

Our special-purpose hardware is a custom wire-wrap board for an IBM personal computer.[4] Our board contains a microprogrammed 7.1 MHz 32 bit finite-state controller and a single 3.6 MHz AMD AmZ8068 DES chip [319]. Data paths 8 bits wide connect the finite state controller, the DES chip, a 16-byte buffer, a PROM computing the next-key function, an 8 bit counter, and the PC Bus interface to the host computer. To increase the board's flexibility, the controller's microprogram is stored in RAM accessible to the host computer. Figure 6.3 shows a simplified block diagram of our special-purpose hardware.

Each algebraic test is programmed in microcode for the board's finite-state controller. The next-key function is computed in a byte-by-byte fashion using a PROM, which can be easily replaced to implement different next-key functions. A read-write counter indicates the number of consecutive messages to compute. By periodically reading the board's counter, the host computer detects completion of the board's activity. Our board also supports all approved modes of operation for DES.

We perform cycle detection in two passes: data acquisition and analysis. During data acquisition, the host computer stores every 2^{20}th message on a floppy disk. During analysis, these messages are loaded into main memory, and up to 2^{20} consecutive messages are computed and compared to those already present. In effect, we perform the Sedgewick-Szymanski algorithm [99] with a fixed estimate of the cycle length. We use an open-addressing, double-hashing scheme for stores and lookups [28]. All data acquisition and analysis routines were written in the C programming

[3] Software implementations of the DES for the IBM PC run at about 200-300 encryptions/second. According to Davio, by using a space-intensive implementation of DES, it is possible to perform about 2.5K encryptions/second on the VAX 11/780 [170]. Thus, it would take the IBM PC about 10 to 16 days to compute 2^{28} DES encryptions; a VAX 11/780 would require about a day and a half. Running the test for 2^{32} steps would take at least 16 times longer.

[4] We chose to use an IBM PC because an IBM PC was available to us, and because it is easy to attach special-purpose hardware to an IBM PC [321].

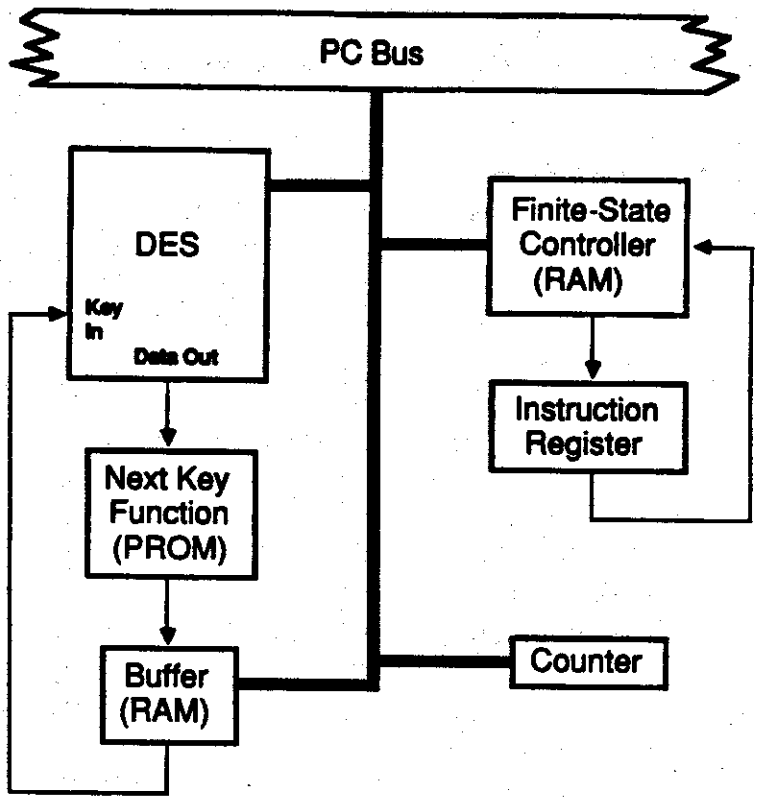


Figure 6.3: Block diagram of special-purpose hardware

language by John Hinsdale, Burton Kaliski, and Ronald Rivest [97].

Using cipher-block chaining in direct control mode [319], the AmZ8068 chip is capable of performing approximately 200K encryptions/second. But for simplicity, we run our experiments using electronic-codebook mode. We clock our control loop at 140ns. and the inner encryption loop at 280ns. Including all overhead for computing and loading a new key for each encryption, our board performs approximately 43K encryptions/second, or about 2^{32} per day. This enables us to carry out each trial of the experiment within a few days.

6.4 Detailed Descriptions of Experiments

This section presents detailed descriptions of the eight cycling experiments we carried out during April to August 1985. The section begins with an explanation of the next-

key functions used in our experiments and of our methods for selecting experimental parameters. The rest of the section consists of nine tables that thoroughly document our experiments.

6.4.1 Notation

In chapter 3, we defined the key space of DES to be the set $\mathcal{K} = \{0,1\}^{56}$. Most DES implementations, however, nominally treat each key as a string of 64 bits, where every eighth key bit is a *parity bit* which is ignored. In this section, we too shall specify keys and messages as 64-bit strings, described in hexadecimal notation. To do this, it is convenient to introduce the DES function $\hat{T}: \hat{\mathcal{K}} \times \mathcal{M} \rightarrow \mathcal{M}$ that operates on the nominal key space $\hat{\mathcal{K}} = \{0,1\}^{64}$.

6.4.2 Next-Key Functions

The cycling closure test depends on a function $\rho: \mathcal{M} \rightarrow \mathcal{K}$ to compute the next key from the current message. We will now describe the two particular *next key functions* that we used during our experiments. We will define each next key function in terms of its related function $\hat{\rho}: \mathcal{M} \rightarrow \hat{\mathcal{K}}$.

Each next key function operated in a byte-by-byte fashion using a byte substitution table (1 byte = 8 bits). For any $0 \leq i \leq 7$ and any $x \in \mathcal{M}$, let $x^{(i)}$ denote the i^{th} byte of x. For each $0 \leq i \leq 7$, we computed $\hat{\rho}(x)^{(i)} = S(x^{(i)})$, for some byte substitution table $S: \{0,1\}^8 \rightarrow \{0,1\}^8$.

In experiments 1 and 2, we chose S to be the identity function. In the other cycling closure experiments, we used the byte substitution table given by table 2.[5] This table was designed so that each entry has odd parity and such that each entry appears exactly twice. The table was generated using the random number generator in the C library on our IBM PC.

For the experiments that used the extended message space $\mathcal{M}^2$, we computed $\hat{\rho}(x)^{(i)} = S(x^{(2i)})$ using the substitution table given in table 2.

6.4.3 Selection of Experimental Parameters

We chose initial messages and keys in a variety of *ad hoc* ways. Some we selected in an obviously deterministic manner (*e.g.*, $x_0 =$ 0123456789ABCDEF). Others are related to our social security numbers or other personal data. The rest we generated using DES and MACSYMA.

[5] The substitution table is used as follows. To substitute any byte B, consider the representation of B as two hexadecimal digits. Select the table entry whose row is given by the first digit and whose column is given by the second digit.

6.4.4 Detailed Experimental Results

The following nine tables give thorough descriptions and results of our cycling experiments. The first table defines the pseudo-random next key function used in several of the experiments. The remaining eight tables—one for each experiment—list all relevant experimental parameters together with important checkpoints encountered during the experiments.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	3E	46	B6	26	AE	F8	2A	AE	CE	57	E6	98	07	5D	92	2C
10	FE	58	EF	CD	F7	76	2F	91	8F	2F	0E	D0	07	B0	73	51
20	20	5E	76	B3	86	9D	16	01	31	EF	D3	8F	D6	40	2A	F8
30	01	C7	C7	19	F7	31	A2	62	9E	B9	DA	D9	34	85	19	D9
40	61	A8	3D	B0	0E	79	C2	BC	52	04	37	FD	6E	85	FB	BA
50	DF	C8	6D	13	43	1C	0B	4A	89	83	E3	20	4F	A7	BA	3B
60	80	D0	67	EA	7F	A8	C8	43	79	6D	1A	4C	A7	CB	86	23
70	5B	02	C2	4C	58	38	FE	CE	B9	1C	15	A4	25	29	1A	15
80	C1	98	7F	4A	64	57	97	32	26	F2	E5	91	D6	E9	6B	F4
90	4F	80	67	DF	F1	BF	B3	B5	3E	E5	7A	EC	A1	B5	92	29
A0	10	DC	97	46	94	CB	49	6B	10	45	3B	F2	E6	FD	B6	BC
B0	40	0D	1F	AD	52	BF	62	23	61	49	E0	0D	08	CD	E3	C4
C0	68	1F	9E	E9	FB	7C	13	75	8A	89	04	5D	6E	DC	54	D5
D0	EA	F1	9D	F4	94	75	D3	70	8C	54	AB	2C	D5	02	98	7A
E0	3D	5B	25	8A	A1	38	8C	EC	70	9B	A4	45	64	51	AB	7C
F0	C1	AD	34	C4	E0	A2	68	83	16	08	DA	32	73	37	0B	5E

Table 6.2: Byte substitution table for pseudo-random next key function.

Experiment 1		
i	x_i	$x_{i+1} = \hat{T}_{x_i}(x_i)$ *Note*
0	0123456789ABCDEF	
34,293,588	B0FDED3BD0DD918C	*end of leader*
34,293,589	AE5530A0E971B5E8	*start of cycle*
2,030,556,568	12B67D3796106D30	*quarter cycle*
4,026,819,547	51AACF5F168E4A17	*half cycle*
6,023,082,526	ED4982C869EF92CF	*three-quarters cycle*
8,019,345,504	A032CE0D3F436EFE	*end of cycle*
8,019,345,505	AE5530A0E971B5E8	*restart of cycle*

Table 6.3: Closure experiment with identity next key function. Cycle length $7{,}985{,}051{,}916 \approx 2^{33}$; leader length $34{,}293{,}589 \approx 2^{25}$.

Experiment 2		$x_{i+1} = \tilde{T}_{x_i}(x_i)$
i	x_i	*Note*
0	121502850B020664	
1,389,523,413	48BB5C9F85CD285A	*end of leader*
1,389,523,414	AFF50E97653421BF	*start of cycle*
5,152,082,299	AE5530A0E971B5E8	*experiment 1 intersection*
9,374,575,329	FB0A1398E92D1473	*end of cycle*
9,374,575,330	AFF50E97653421BF	*restart of cycle*

Table 6.4: Closure experiment with identity next key function. Cycle length $7{,}985{,}051{,}916 \approx 2^{33}$; leader length $1{,}389{,}523{,}414 \approx 2^{30}$.

Experiment 3		$x_{i+1} = \tilde{T}_{p(x_i)}(x_i)$
i	x_i	*Note*
0	6036222982B03104	
2,138,241,978	68955F4BF000A6E0	*end of leader*
2,138,241,979	C9DB8E7169CCF272	*start of cycle*
3,706,679,992	433B74E2CB18DDFD	*end of cycle*
3,706,679,993	C9DB8E7169CCF272	*restart of cycle*

Table 6.5: Closure experiment with pseudo-random next key function. Cycle length $1{,}568{,}438{,}014 \approx 2^{30.5}$; leader length $2{,}138{,}241{,}979 \approx 2^{31}$.

Experiment 4		$x_{i+1} = \tilde{T}_{p(x_i)}(x_i), x_i \in \mathcal{M}^2$
i	x_i	*Note*
0	4C957F303AC4D08B 63E15C9C7A398042	
4,294,967,296	2C173869EAF8804B 767469BB19B26D8A	2^{32} *iterations*
8,589,934,592	4349368A49700D3B 55FC02F8848BC64F	2^{33} *iterations*
12,884,901,888	55D1292F5D99B268 C30AB80FF3B03D08	$3 \cdot 2^{32}$ *iterations*
17,179,869,184	4A224C65B8A48DEB 00C7D0CA64C4B240	2^{34} *iterations*

Table 6.6: Extended closure experiment with pseudo-random next key function. No cycle detected in 2^{34} steps.

Experiment 5	$x_{i+1} = \hat{T}_{\hat{k}}^{-1}\hat{T}_{\hat{p}(x_i)}(x_i)$	
i	x_i	*Note*
0	0123456789ABCDEF	
3,233,340,362	0EC45F7157BD8749	*end of leader*
3,233,340,363	EFE7B7112233DD88	*start of cycle*
4,531,729,424	C09DFA478C3849BE	*end of cycle*
4,531,729,425	EFE7B7112233DD88	*restart of cycle*

Table 6.7: Purity experiment with pseudo-random next key function. Cycle length $1{,}298{,}389{,}062 \approx 2^{30}$; leader length $3{,}233{,}340{,}363 \approx 2^{31.5}$. Key $\hat{k} =$ 97778E1BC3FD8E07.

Experiment 6	$x_{i+1} = \hat{T}_{\hat{k}}^{-1}\hat{T}_{\hat{p}(x_i)}(x_i)$	
i	x_i	*Note*
0	12150285080020664	
1,366,287,307	E43D6EF9351DDB4A	*end of leader*
1,366,287,308	75C6C23C21EA50DA	*start of cycle*
5,584,675,814	FDBE1ECDF38BF3E5	*end of cycle*
5,585,675,815	75C6C23C21EA50DA	*restart of cycle*

Table 6.8: Purity experiments with pseudo-random next key function. Cycle length $4{,}218{,}388{,}507 \approx 2^{32}$; leader length $1{,}366{,}287{,}308 \approx 2^{30}$. Key $\hat{k} =$ 4D3FD0FED9A4FA9B.

Experiment 7	$x_{i+1} = \hat{T}_{1\ldots1}(\hat{T}_{0\ldots0}(x_i))$	
i	x_i	*Note*
0	0123456789ABCDEF	*start of cycle*
2,227,161,945	654B672D3DBC73AB	0...0 *fixed point*
4,454,323,890	293FD4F2C13DD94F	*"hidden crossing"*
5,890,012,565	3CC5B06ADEFD30A0	1...1 *fixed point*
7,325,701,239	0123456789ABCDEF	*restart of cycle*

Table 6.9: Orbit experiment using composition of weak keys. Cycle length $7{,}325{,}701{,}239 \approx 2^{33}$; leader length 0.

Experiment 8		$x_{i+1} = \hat{T}_{\hat{k}}(x_i)$
i	x_i	*Note*
0	41184DCAB17324C8	
17,179,869,184	B98C3A67CD6F8267	2^{34} *iterations*
34,359,738,368	632509BC9F57DF8A	2^{35} *iterations*
51,539,607,552	ED4B06ABBF5515FB	$3 \cdot 2^{34}$ *iterations*
68,719,476,736	2C84263510AEAD34	2^{36} *iterations*

Table 6.10: Orbit experiment. No cycle detected in 2^{36} steps. Key $\hat{k} =$ 116E0B8278AEC431.

Chapter 7

Open Problems

This chapter outlines two directions for further research inspired by part I. Section 7.1 describes several open questions about the algebraic structure of DES. Section 7.2 states formally the problem of testing cryptosystems for algebraic structure and raises questions about the complexity of this problem.

7.1 Open Questions about DES

Although our experiments give strong statistical evidence that DES is not closed, numerous questions remain unanswered. This section lists several open questions about the algebraic structure of DES.

- Does DES generate $\mathcal{A}_M$? What is the order of the group generated by DES? What is the group generated by DES? For how many keys i, j, k is it true that $T_k = T_i T_j$?
- Is DES faithful? What is the order of DES?
- What subsets of DES transformations generate small groups? (Note that each weak key represents a transformation that generates the cyclic group of order 2.)
- Is DES *homogeneous* in the sense that for every $k \in \mathcal{K}$ it is true that $T_k^{-1} \in \mathcal{T}_{DES}$? For how many $k \in \mathcal{K}$ is it true that $T_k^{-1} \in \mathcal{T}_{DES}$?
- Is $I \in \mathcal{T}_{DES}$?

Our results show that the composition of every pair of weak keys will likely have a short orbit for every message. It would also be interesting to know if other special pairs of DES transformations have similar properties. For example, it would be interesting to explore semi-weak keys, *light keys* (keys with a low density of ones), *heavy keys* (keys with a high density of ones),

and pairs of related keys (*e.g.* keys that differ in exactly one bit and keys that are complements of each other).

Knowing whether or not $I \in \mathcal{T}_{DES}$ is interesting—not because this property would necessarily be a weakness in DES—but because this question would answer several other questions about DES. By the complementation property, for any key k, $T_k = I$ implies $T_{\bar{k}}$=I. Hence, if $I \in \mathcal{T}_{DES}$, then DES is not faithful. In particular, if DES is closed, then DES is not faithful. Conversely, if $I \notin \mathcal{T}_{DES}$, then DES is not closed.

Each of the known-plaintext attacks finds a representation of the secret transformation T_k as a product of two or more transformations. In practice, it would suffice to find an approximate representation of T_k. To this end, we could say that two permutations $T_1, T_2 \in \mathcal{T}_{DES}$ are *q-approximately equal on* $X \subseteq \mathcal{M}$ if and only if, for all $x \in X$, $T_1(x)$ and $T_2(x)$ always agree on at least q bits.

- For each $1 \leq q \leq 64$, for how many keys i, j, k is it true that T_k is *q-approximately equal* to $T_i T_j$ on $\mathcal{M}$?
- What other notions of "approximately equal" transformations would be useful in finding approximate representations?

Since the closure tests do not depend on the detailed definition of DES, it is natural to ask:

- What can be proven from the detailed definition of DES about the order of the group generated by DES?
- Are there more powerful statistical closure tests than the two tests presented in this paper that are based on the detailed definition of DES?

Our research also raises questions involving the design of cryptosystems.

- Is it possible to build a secure, practical cryptosystem for which it can be proven that the cryptosystem generates either $\mathcal{A}_{\mathcal{M}}$ or $\mathcal{S}_{\mathcal{M}}$? (See [164] for one suggestion.)
- Is it possible to hide a trapdoor in a cryptosystem by concealing a secret set of generators for a small group? (Note that it does not work simply to have a large subset of the transformations generate a small group, since the enemy could guess a small number of transformations in the subset and apply the cycling closure test to the guessed transformations.)

We presented two known-plaintext attacks against closed ciphers, but other attacks may also exist.

- What attacks are possible against closed ciphers? How can knowledge of the specific group help?

Finally, it would be interesting to apply the closure tests to variations of DES that exaggerate certain types of possible weaknesses in the standard.

- What is the order of "crippled" DES transformations formed by reducing the number of rounds or by replacing one or more of the S-boxes with linear mappings?

7.2 Complexity of Detecting Algebraic Properties

The cycling closure test runs sufficiently fast that we were able to apply it to DES using an IBM PC equipped with a custom board. But is the cycling test the most efficient algorithm for determining if an indexed set of permutations forms a group under functional composition? This section takes a modest step toward answering this question by formulating the problem of testing an indexed set of permutations for closure and by discussing some of the issues involved with determining the complexity of this problem.

7.2.1 Group Detection Game (GDG)

The problem of determining if an indexed set of permutations forms a group under functional composition can be stated in terms of a two-person game, which we shall call the *Group Detection Game (GDG)*. Let Alice and Bob be the two players in this game. To start the game, Bob selects a group G. Then, by means of a "black box," Bob gives Alice either G or a set of randomly chosen set of permutations. Alice must decide if the black box computes the group. Bob selects the group to make Alice's task as hard as possible.

Three parameters u, v, m control the game. The parameters u, v specify a *message space* $\mathcal{M} = \{0,1\}^u$ and a *key space* $\mathcal{K} = \{0,1\}^v$; the parameter $m \leq 2^v$ specifies the order of the group. Let $M = 2^u$ and $K = 2^v$. At the beginning of the game, both players know u and v, but only Bob knows m.

The game consists of three moves, the last two of which may be repeated several times. First, Bob selects a particular permutation group G acting on $\mathcal{M}$ of order m. Second, Bob gives Alice either a "random presentation of G" or a random set of m permutations acting on $\mathcal{M}$. Bob's gift to Alice takes the form of a "black box" oracle, and Bob decides what to give Alice at random. Third, after experimenting with the box, Alice outputs "Group," if she believes Bob gave her G; otherwise, Alice outputs "Random."

The *black box* oracle computes an indexed set $\mathcal{T} = \{T_k\}_{k \in \mathcal{K}}$ of permutations on $\mathcal{M}$. Given any key $k \in \mathcal{K}$ and any message $x \in \mathcal{M}$, the box computes $T_k(x)$ and $T_k^{-1}(x)$. Initially Alice has no other information about the box; the only way Alice can learn more about the box is through asking it to compute $T_k(x)$ and $T_k^{-1}(x)$ for specific k's and x's.

Let $G = \{g_1, \ldots, g_m\}$ be the group chosen by Bob. We will now explain what is meant by a *random presentation of* G. Whenever Bob gives Alice a random presentation of G, the black box computes the function $T : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{M}$ defined by $T(k, x) = g_{\pi(k)}(x)$, for all $k \in \mathcal{K}$ and all $x \in \mathcal{M}$, where $\pi : \mathcal{K} \rightarrow \{1, \ldots, m\}$ is a randomly chosen function unknown to Alice. If steps 2 and 3 of the game are repeated, then Bob chooses the function π independently for each repetition of step 2.

Many variations of this game are possible. For example, Bob might give Alice some "side information" about G, such as m or a range in which m lies. Alternately, Bob might be required to pick G of some special form (*e.g.* an alternating group).

Let p_A denote *Alice's success rate*—the chance, on any repetition of steps 2 and 3, that Alice correctly guesses whether Bob gave her a presentation of G. By guessing randomly, Alice can achieve a success rate of $p_A = 1/2$. By applying the cycling closure test and using $O(\sqrt{K})$ black box queries, Alice can achieve a success rate approaching 1.

Motivated by the DES example, we are interested in the situation in which M and K are so large that both $O(M)$ and $O(K)$ are "intractable" running times, yet such that $O(\sqrt{M})$ and $O(\sqrt{K})$ are "tractable." We also assume that $M^l > K$ for some small integer l.

The major open problem of this section can now be stated as follows: given any $0 < \epsilon \leq 1/2$, how many black box queries must Alice make in order to achieve a success rate of at least $1/2 + \epsilon$?

7.2.2 Discussion

We conclude this section with a brief discussion of some of the issues that arise in the group detection game. These issues include certain crucial elements of the model, Bob's choice of group, and what side information, if any, Bob gives Alice. We leave as an open problem a complete analysis of how these issues affect Alice's optimal strategy.

To make Alice's task difficult, it is important for Bob to choose a group that "looks random." Unless the group looks random, Alice can detect the group by identifying structural properties. For example, Alice can easily detect any abelian group by testing for commutativity. It would be interesting to know what other algebraic properties are useful and possible for Alice to detect. Babai conjectures that, for small l, Alice can easily test for solvability of length l and nilpotence of class l and hence advises Bob against choosing a group with these structures. But

what group should Bob pick? Perhaps an alternating group would complicate Alice's task the most.

In essence, Alice's task is to distinguish structure from lack of structure. One tool at her disposal—perhaps her only tool—is to look for "identities" among the indexed permutations. For example, every commutative group satisfies the identity $\alpha\beta = \beta\alpha$, for all group elements α and β. Alice can look both for identities that hold for all of the group elements as well as for identities that hold only for some fraction of the group elements, or even for some fraction of the group elements on some subset of the message space. Bob should choose a group for which it is very time consuming to find any nontrivial identities.

Any side information given to Alice that reveals algebraic properties of the group might help Alice discover useful identities. In particular, the order of the group reveals much information about the group.

One nuisance for Alice is that the group might be represented unfaithfully (*i.e.* two different keys might represent the same permutation). This possibility complicates some of the algebraic tests Alice might wish to perform.

The orbit test, which searches for short orbits of particular messages on particular permutations, can be viewed as a test that looks for identities. The strength of this test depends on Bob's choice of group. It would be interesting to analyze the orbit test further, beginning with its strength on alternating groups.

In analyzing the orbit test, it is useful to know the distribution of the order of elements in various types of groups. Erdõs and Turan [90] and Bovey [87] studied this problem for the symmetric group. For any n, let S_n denote the symmetric group on n letters. Moreover, for any $g \in S_n$, let $\xi_n(g) = \log_e \text{order}(g)$ be the natural logarithm of the order of g in S_n. Erdõs and Turan proved, that for large n, ξ_n has a normal distribution with mean $(\log_e n)^2/2$ and standard deviation $(\log_e n)^{3/2}/\sqrt{3}$. Thus, for any technology bound B, for the expected order of an element drawn at random from S_n to be at most B, it must be true that $n \leq e^{\sqrt{2 \log_e B}}$.

One important aspect of our model concerns how much time is required to compute compositions of groups elements. In our model, for any message x, for any group element g, for any integer n, it requires n black box queries to compute $g^n(x)$. In light of the orbit test, different assumptions about the complexity of computing compositions can significantly affect the analysis of the group detection problem.

One strategy for Alice is to apply the cycling closure test. There are some reasons to suspect that the $O(\sqrt{K})$ time required for this test is approximately the best Alice can do. The intuition goes as follows. Unless Alice discovers any nontrivial identity, Alice cannot do better than random guessing. Moreover, since Alice is given a random presentation of the group, the Birthday paradox implies that it is extremely unlikely for Alice to discover a nontrivial identity unless she examines at least $\sqrt{K}$ permutations.

Another promising strategy for Alice is to perform a case analysis based on m,

the order of the group. Even if Alice does not know m, she can investigate various hypotheses about m. For example, if m is prime, then the group must be cyclic. If m is odd, then the group must be solvable. If m is the product of exactly l prime factors (not necessarily distinct), then the group is solvable of length at most l. If $m = 2^k$ for some k, then the group is nilpotent of class k. Other conditions imply that short orbits are easy to find. And so on. For each hypothesis, Alice can perform an algebraic test. Whether or not such an approach would be more effective than the cycling closure test, we leave as an open problem.

Chapter 8

The PI Project

Layout efficiency affects the cost and performance of VLSI chips. Despite the complexity and importance of finding efficient layouts, this task is usually carried out in part by humans following *ad hoc* techniques. The MIT PI System offers an algorithmic approach for automating the layout process.

The second part of this dissertation deals with the problem of placing modules on a VLSI chip, focusing on the placement algorithms used in the PI System. PI's placement algorithms are built around a recursive mincut strategy that is sensitive to geometric and graph-theoretic concerns. Supported by a data structure known as the *placement tree*, these algorithms are implemented in a general framework in which approximate and partial placements can be represented and manipulated.

During his involvement with the PI Project, Alan Sherman designed and implemented the placement algorithms and participated in the design of many other parts of the PI System.

The *PI (Placement and Interconnect) Project* explored algorithmic approaches for laying out VLSI chips. Work on the PI Project centered around the design and implementation of the *PI System*, one of the first fully automatic systems for laying out custom VLSI chips. A characteristic feature of the PI System is its problem decomposition, which divides the layout process into separate placement and routing phases each of which are further divided into subproblems and solved by specialized component algorithms.

The PI Project was carried out at the MIT Laboratory for Computer Science under the leadership of Professor Ronald L. Rivest. Initial motivation for the project grew out of Rivest's experiences implementing the RSA cryptosystem on a chip [227]. Recognizing the need for better layout tools, Rivest, Adleman, and Shamir wrote several programs to help them lay down the wires on their first RSA chip. Following the completion of this RSA *n*MOS chip, work on the PI Project began in early 1981 and continued through 1983. During this time, the PI Project identified important layout problems, designed heuristics for solving these problems, and produced an

initial implementation of the PI System. Over the course of the project, a total of approximately twenty undergraduate and graduate students contributed to the design and development of PI.[1]

This chapter introduces part II through giving an overview of the PI System, including an example of how PI lays out a small chip.

8.1 Overview of the PI System

This section gives a brief overview of the PI System, concentrating on its input/output behavior and on its overall organization.

Input to PI describes a circuit to be be laid out. The input consists of a set of arbitrarily shaped rectangular modules and a set of nets that specify how the modules are to be interconnected. Output from PI gives a complete, detailed layout of the circuit. The output specifies an exact, nonoverlapping placement of the modules in a rectangular region, together with a detailed plan of where each wire interconnecting the modules should be laid. Subject to the Mead-Conway style design rules for single metal layer CMOS or *n*MOS designs [341], PI attempts to minimize total chip area and the amount of wire used for routing.

PI partitions the layout process into a series of subproblems and solves each subproblem using a specialized component algorithm. After checking its input, PI lays out the specified circuit in four major steps: module placement, power-ground routing, signal routing, and compaction. Each of these steps is further decomposed into additional subproblems. For example, signal routing is accomplished by channel definition, global routing, crossing placement, and switch-box channel routing. Figure 8.1 summarizes this process.

After module placement, two problems can arise both of which are solved by PI's resizer. First, if the placement does not leave enough routing space, then there might not be any legal way to complete the layout. Second, even if some legal layout exists, the routers might not be clever enough to find a routing. Whenever routing fails, the resizer is called to expand congested areas of the chip. Then routing is reattempted. The resizer also performs the final compaction step, which attempts to squeeze out any unnecessary remaining space. Thus, the resizer plays a crucial role in the overall structure of PI.

Most of PI's algorithms are heuristic in nature. The decision to seek heuristic solutions arose from the fact that most problems encountered by PI are NP-complete. Some of PI's algorithms perform optimally in certain special cases, and some of the algorithms are provably good approximation algorithms. But most of PI's algorithms

[1] See section 15.2.2 for a description and acknowledgment of the contributions made by members of the PI Project.

1. Input problem and check input
2. Place modules
3. Route power and ground wires
   (a) Lay power and ground rings
   (b) Route ground tree*
   (c) Grow power forest*
   (d) Stretch power and ground wires to meet current requirements
4. Route signal wires
   (a) Define channels
   (b) Route signal nets globally*
   (c) Determine crossing placements
   (d) Route channels*
5. Compact layout
6. Output description of chip

**Note: If routing fails at any of these steps, the resizer is called to expand the layout, and routing is reattempted.*

Figure 8.1: Outline of PI System

come with no formal proof of performance. Nevertheless, PI's algorithms are based on sound principles and yield good results in practice.

All parts of the PI System have been implemented, except for the resizer. Although it was never a goal of the PI Project to produce a production-quality implementation of the PI System, the initial implementation of the PI System served a useful role in assessing the feasibility of PI's approach and in assuring that every detail had been accounted for.

8.2 How PI Lays Out a Chip: An Example

This section illustrates PI's step-by-step performance on a small example. Created by PI's random example maker, the example involves 10 logic modules, 10 pads, and 7 signal nets. PI laid out this chip in March 1984.

Pictures in this section are photocopies of color prints made from 35mm color slides taken of the screen of a Symbolics 3600 Lisp Machine. Modules are displayed as black rectangular regions. However, if two or more modules overlap, then the following exclusive-or display rule is used: regions with an even number of overlapping modules are displayed in white, and regions with an odd number of overlapping modules are displayed in black.

PI first estimates chip size and then places the pads around the chip's periphery. In this example, to minimize chip area, PI placed the pads on only two (opposite) edges of the chip (see figure 8.2). PI arranges the pads using a heuristic that brings together pads that are strongly connected through the circuit. After pad placement, PI temporarily places all of the logic modules on top of each other at the origin of the *logic box*, the central rectangular region allocated for placing the logic modules (see figure 8.2).

Following pad placement, PI builds an approximate placement using a top-down recursive mincut heuristic. Figures 8.3–8.5 illustrate this process. At each step, PI partitions the modules from the current region into two subsets, each approximately with the same total module area. A graph partitioning heuristic is used to bring together modules that are highly connected to each other. Corresponding to the partitioning of the modules, PI also slices the current rectangle into two rectangles. The modules associated with each of the newly created rectangles are displayed at the origin of their associated rectangle. PI tries both horizontal and vertical partitions, and selects the one it deems best as measured by a score function. PI takes into consideration both the number of nets that must cross the partition, the balance of module areas on each side of the partition, and the aspect ratios of the resulting rectangles.

Continuing in a breadth-first fashion, the mincut process builds a *placement hierarchy*. This hierarchy specifies a recursive partitioning of the logic modules together

with a recursive slicing of the logic box. The mincut process terminates when each module is associated with a unique leaf rectangle in the recursive slicing (see figure 8.6). After the mincut phase, the resulting placement is still approximate—the modules have not yet been flipped or rotated, and the modules might even overlap since they might not fit into their associated rectangles.

To transform the approximate placement produced by mincut into an exact, legal placement, PI undertakes a recursive bottom-up process called *hardening*. During this process, PI successively "glues" modules and "super modules" together working its way up the placement hierarchy.[2] At the leaf level, each module is flipped and rotated to minimize chip area and estimated wire length. As each pair of modules is glued together, PI leaves space for routing and aligns the modules to facilitate routing. After a pair of modules is glued together, PI creates a minimum bounding box around the pair. This newly created box then becomes a "super module." In the hardening process, supermodules are treated like modules, except that they are neither flipped nor rotated.

Figures 8.7–8.9 show three successive steps in the hardening process. In figure 8.7, PI has just finished gluing the second pair of modules together. Specifically, in figure 8.7 modules A and B have been glued together and modules C and D have been glued together. In figure 8.8, PI has glued the super module consisting of modules C and D with module E. In figure 8.9, PI has glued the super module consisting of modules A and B with the super module consisting of C, D, and E. These pictures can be confusing to interpret since PI redisplays the newly created boxes on top of the approximate placement produced by mincut.

Figure 8.10 shows the exact placement produced by the hardening process. Figure 8.11 shows the same placement without the placement hierarchy. This picture marks the end of PI's placement phase.

After placing the modules, PI routes the power and ground nets in three steps. First, PI lays a ground ring around the outside of the pads and PI grows a ground tree inside the chip. To keep the ground tree concentrated in the center of the chip, PI uses a novel heuristic based on Hamiltonian circuits (see figure 8.13). Second, PI lays power wires between the pads and the logic modules and PI grows a power forest to supply power to the logic modules (see figure 8.14). During power-ground routing, PI enlists the help of the channel routing routines. For example, before laying the ground tree, PI divides the routing area into rectangular channels (see figure 8.12). Not shown in this example is the final step of power-ground routing in which PI calls the resizer to stretch the power and ground wires to meet their current carrying requirements.

Figure 8.15–8.17 show how PI routes the signal nets. To begin, PI redefines the channels after power-ground routing (see figure 8.15). Regions under the metal power

[2] The reader may find it helpful to think about the TEX text formatting system in which the "glue" between boxes can be set.

and ground wires are treated as special *covered channels* in which highly restrictive routing rules apply.

Using a Steiner tree heuristic, PI routes each signal net in a coarse, global fashion (see figure 8.16). Then, PI enters a novel stage known as *crossing placement*: for each channel edge, PI determines the exact layer and position at which each net crosses the edge. During crossing placement, PI attempts to minimize globally the number of times different nets must cross each other. The crossing-placement stage is not shown graphically.

Next, PI independently routes each of the fixed-sized switch-box channels.[3] For each channel, PI applies up to three different channel routers, one after the other, before succeeding or giving up. PI's channel routers consist of a pattern router, a greedy left-to-right slice router, and a Lee router based on a shortest path algorithm. Figure 8.17 shows the detailed routing of all nets together with the channel definitions. Small dark squares indicate contact cuts, where wires on different layers connect. In this example, all but two tiny channels between the upper pads and the power wire routed successfully.

At this point, PI calls the resizer to expand the two channels that failed to route successfully. After all channels are successfully routed, PI calls the resizer one last time to compact the layout. Including display time, PI took about five minutes to lay out this example.

[3] A *switch-box* channel is a rectangular channel that may have wires entering or leaving any or all sides of the channel.

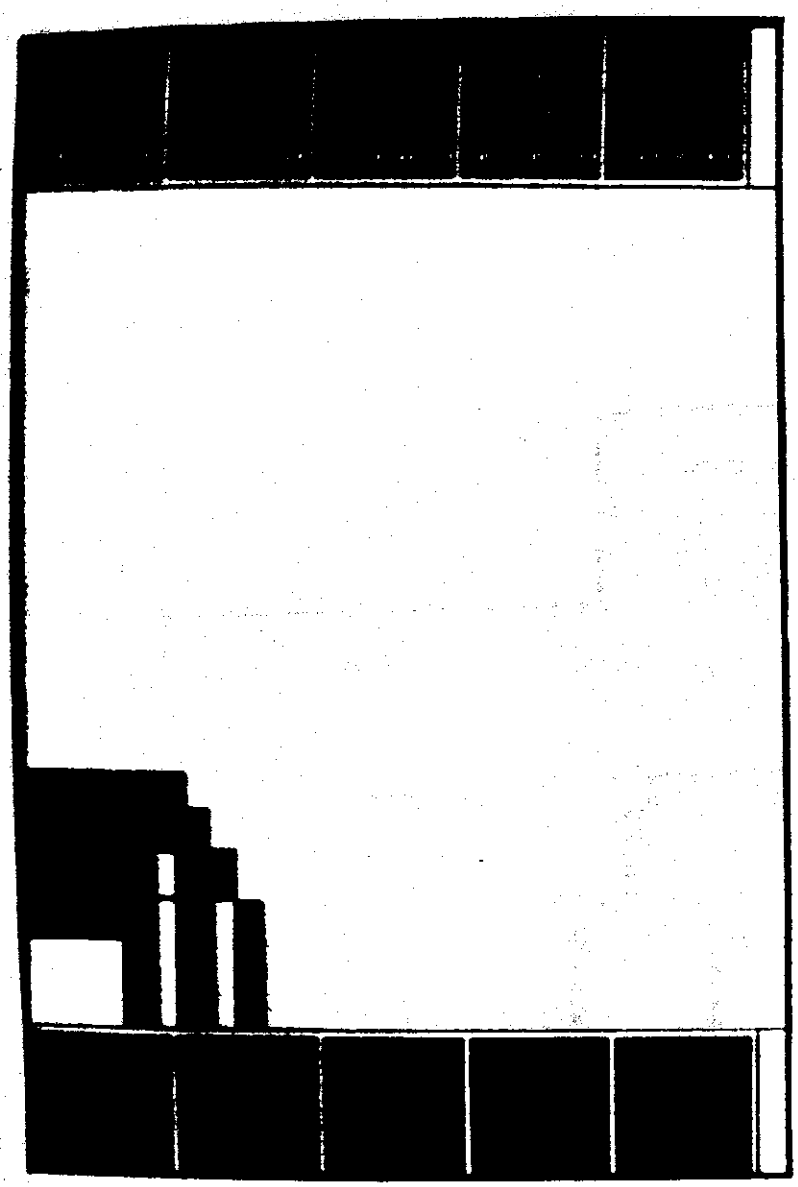


Figure 8.2: Pad placement

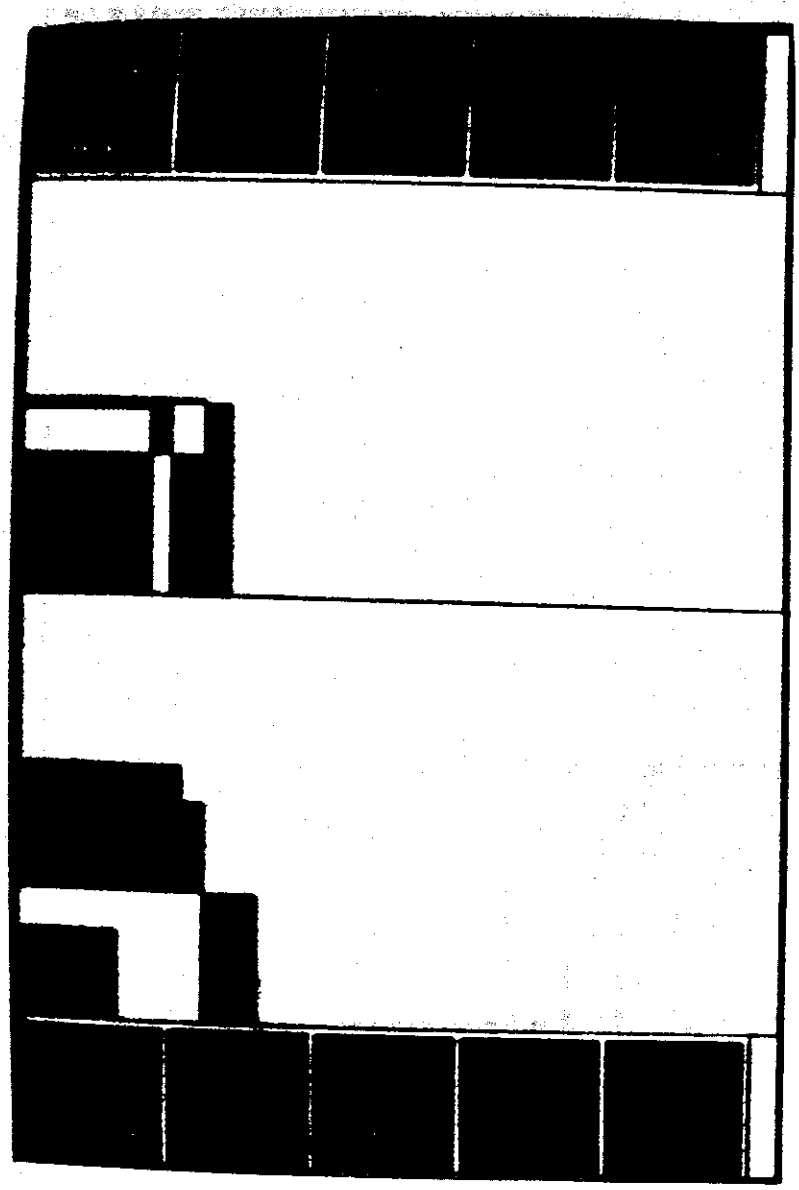


Figure 8.3: Building the approximate placement: Mincut step A

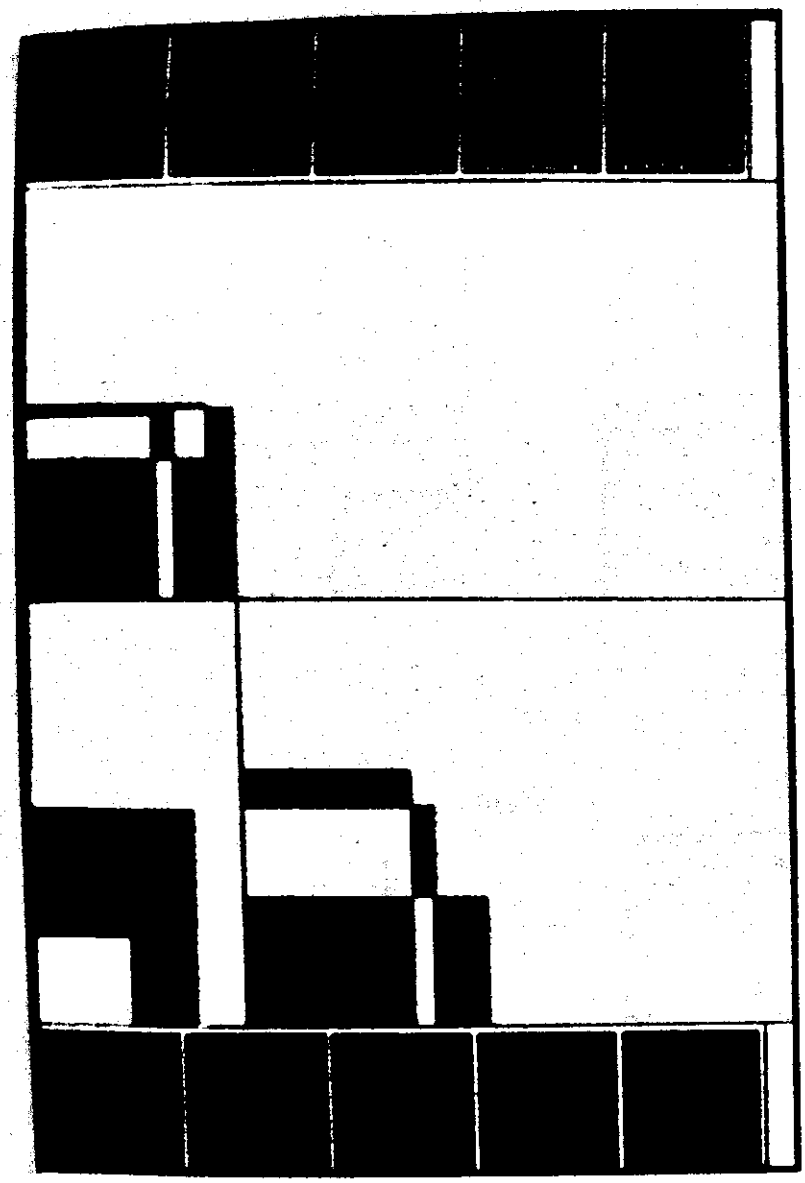


Figure 8.4: Building the approximate placement: Mincut step B

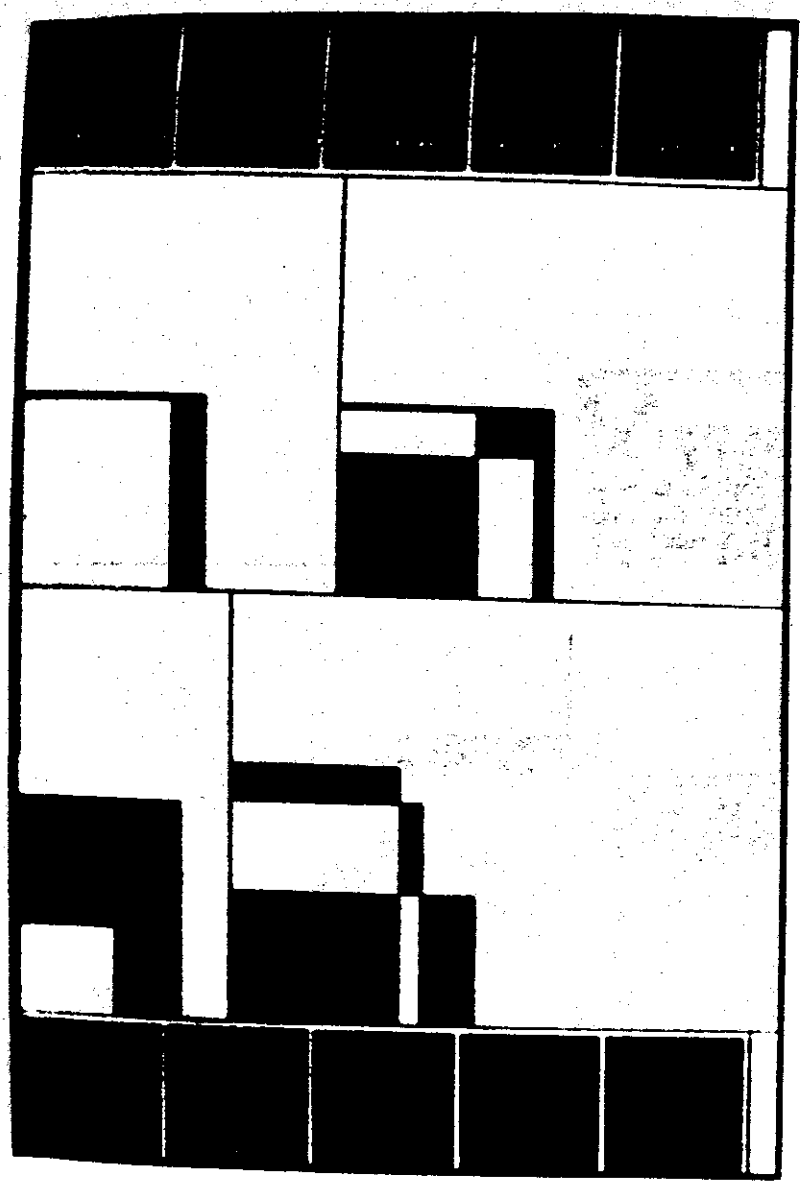


Figure 8.5: Building the approximate placement: Mincut step C

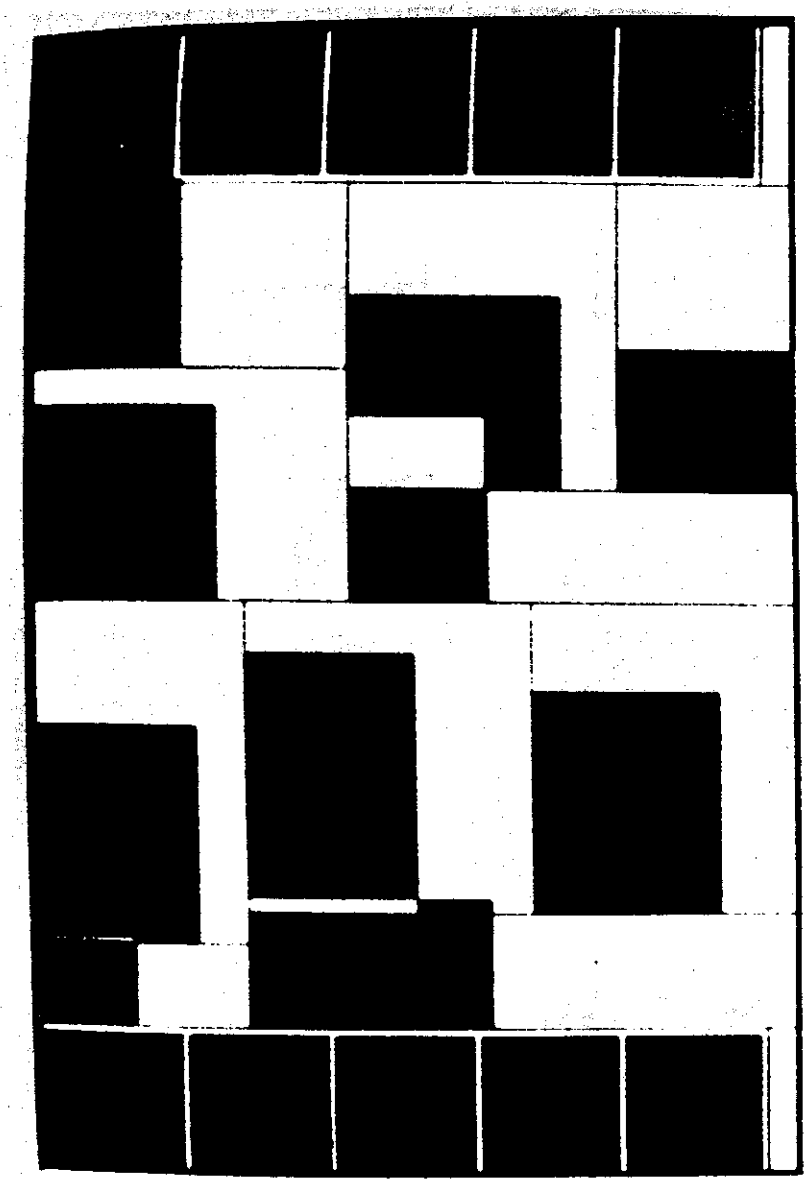


Figure 8.6: Approximate placement after mincut

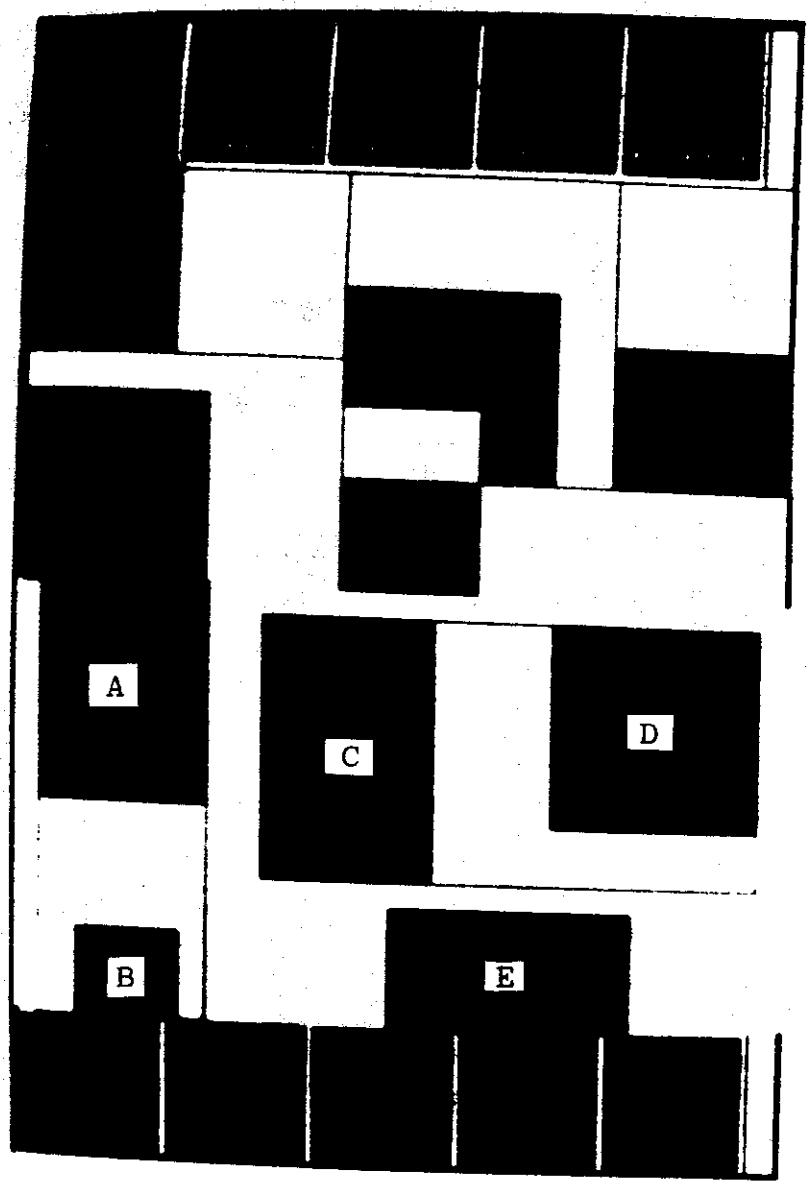


Figure 8.7: Determining exact placement: Hardening step A

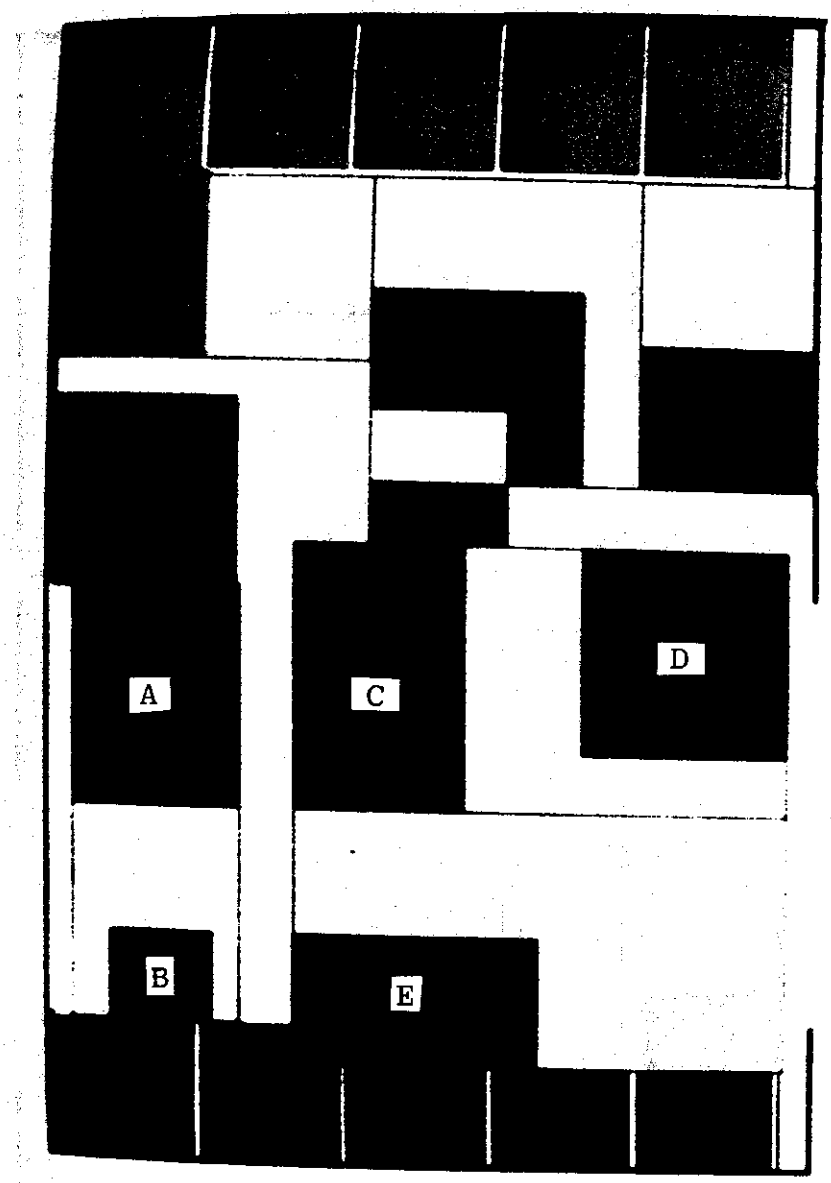


Figure 8.8: Determining exact placement: Hardening step B

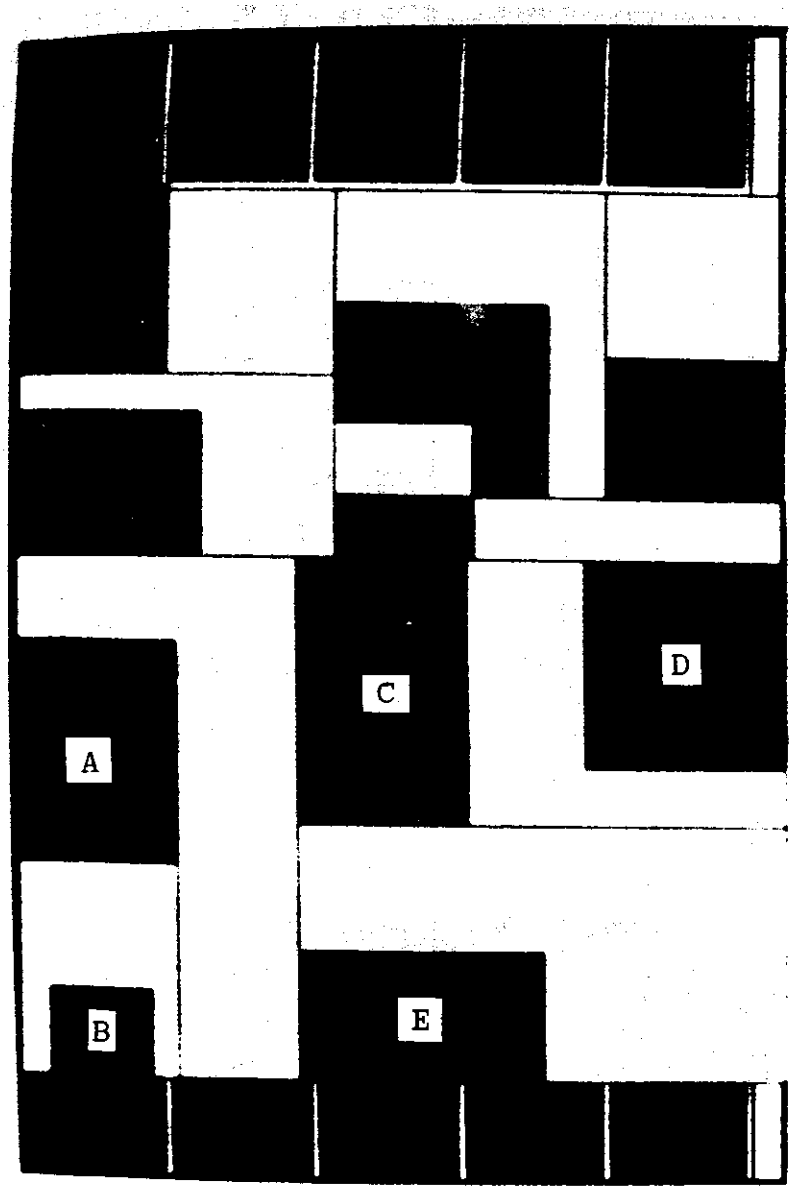


Figure 8.9: Determining exact placement: Hardening step C

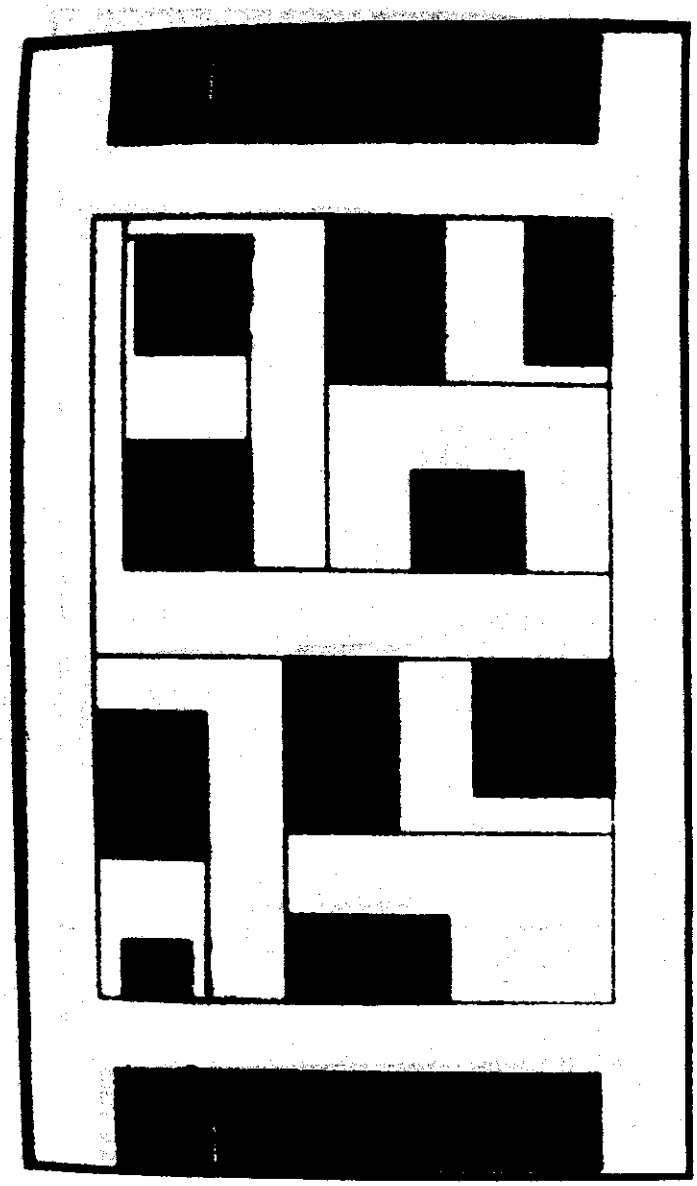


Figure 8.10: Exact placement with placement hierarchy

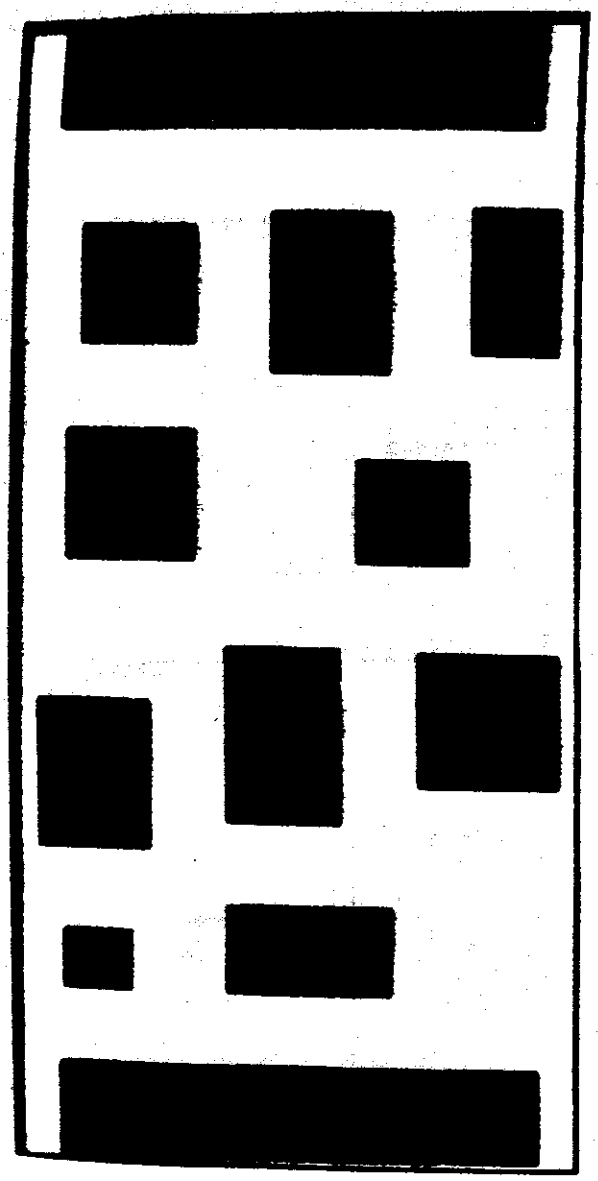


Figure 8.11: Module placement

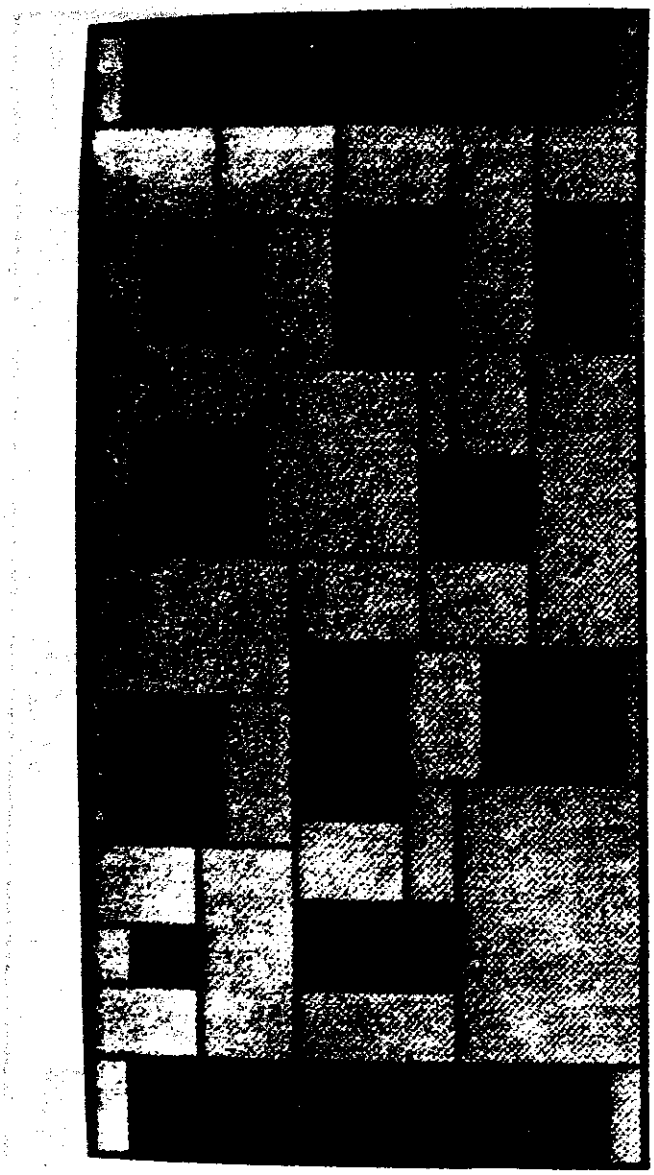


Figure 8.12: Channel definition before power-ground routing

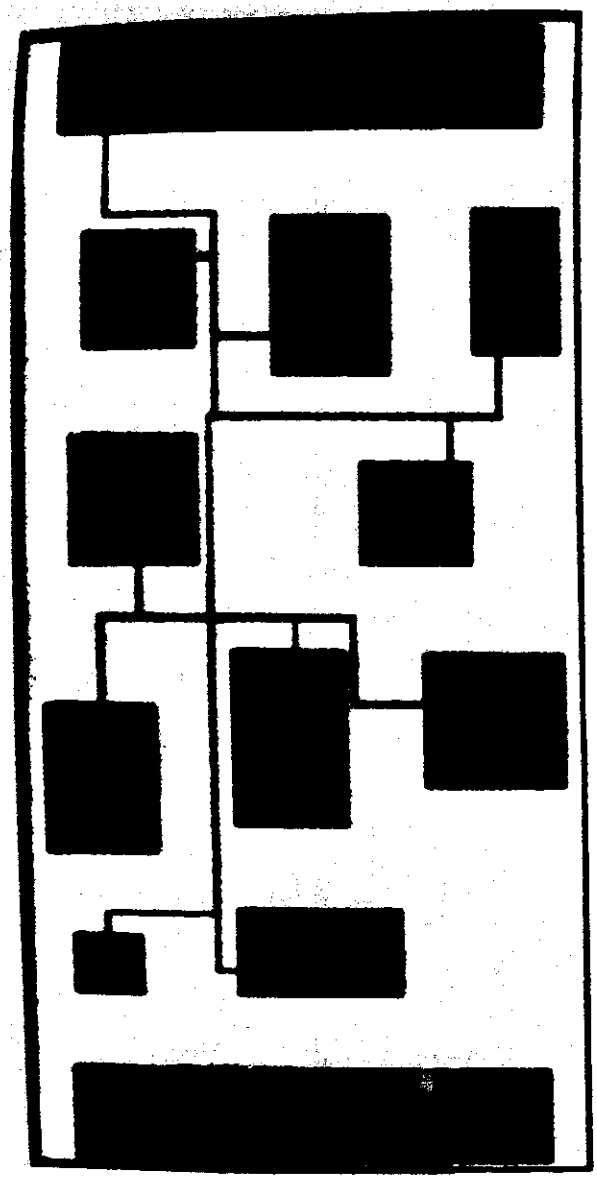


Figure 8.13: Routing of ground tree

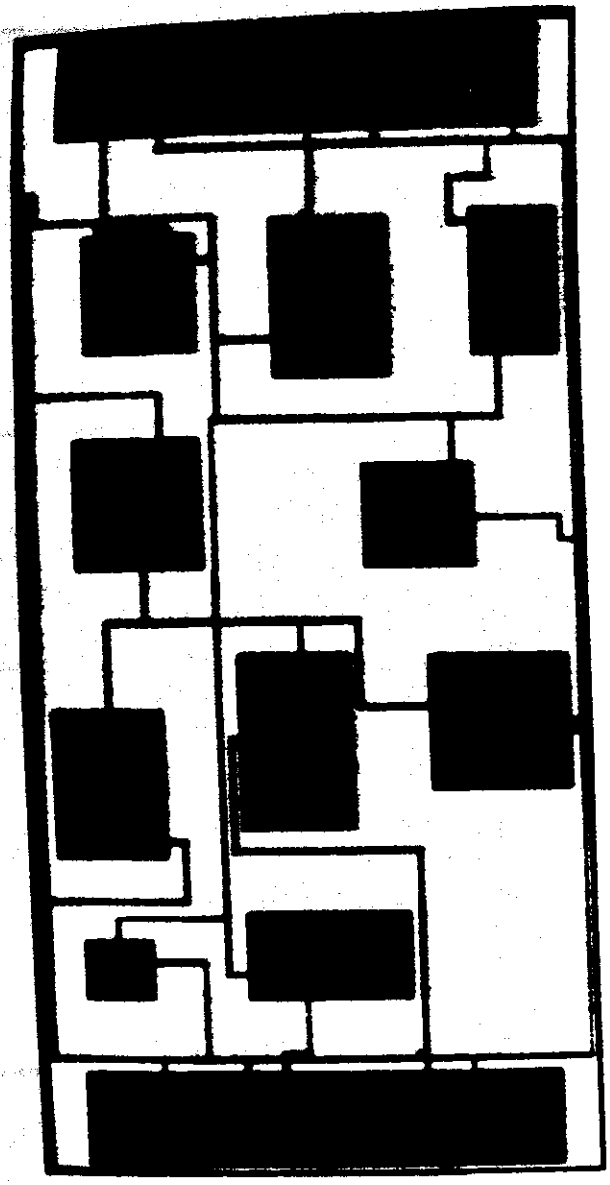


Figure 8.14: Routing of power forest

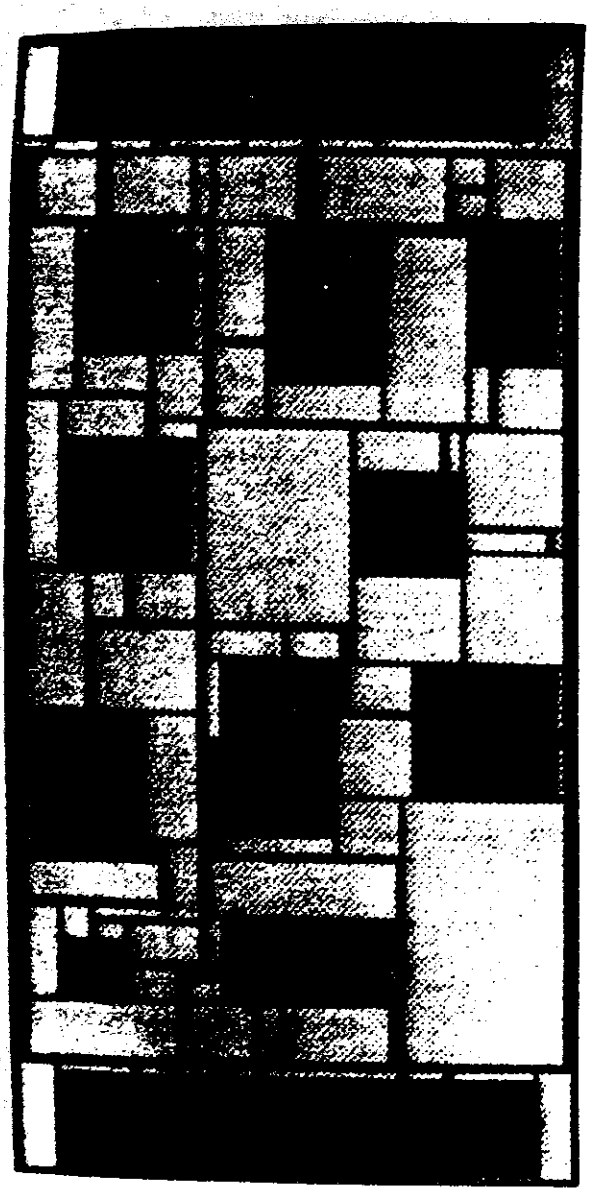


Figure 8.15: Channel definition after power-ground routing

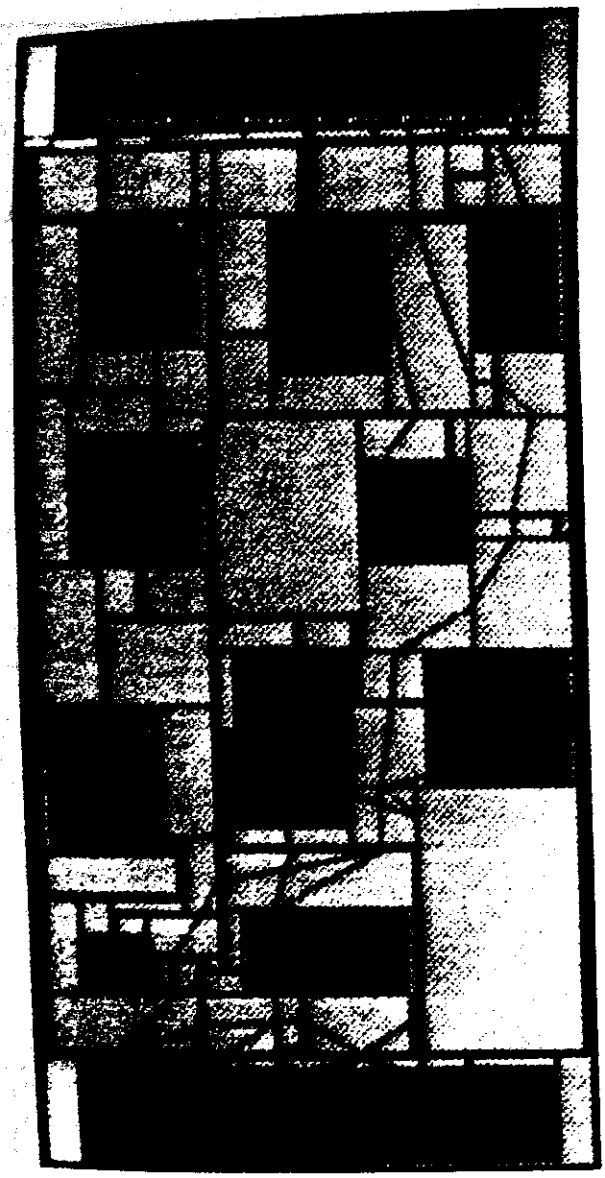


Figure 8.16: Global routing of signal nets

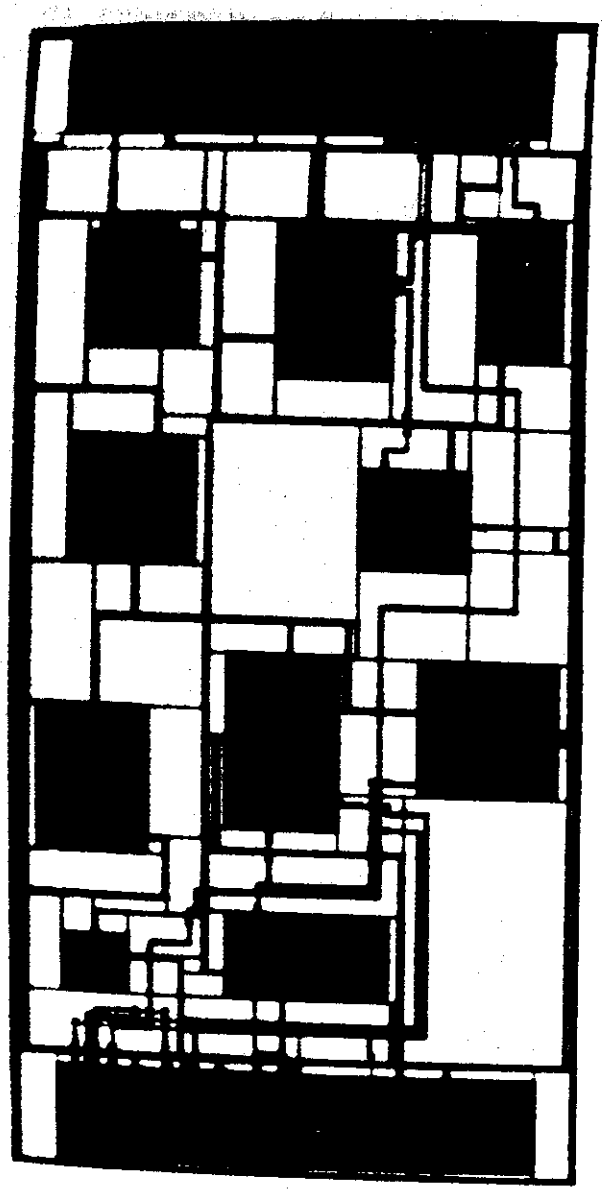


Figure 8.17: Layout after channel routing of signal nets

Chapter 9

Preliminaries

This chapter presents additional background about the PI System helpful in understanding PI's placement algorithms. The chapter also explains the notation and terminology used throughout part II of this dissertation.

9.1 The PI System

This section describes the PI System in more detail, focusing on its objectives, input/output specifications, modes of operation, layout model, major design decisions, and layout representation.

9.1.1 Objectives

The major goal of the PI System was to design and implement an automatic method for laying out custom VLSI chips. PI was not supposed to be an intelligent layout editor, but rather an automatic layout system, which, with a minimum of human interaction, would quickly produce high-quality results. It was intended that PI might eventually be used for such applications as laying out a chip that was designed by a student as a course project. Such a chip might consist of, say, a few dozen modules and a few hundred nets. PI was to work quickly enough to lay out such a chip in at most a few hours, and the results were to be good enough to send for fabrication. PI was intended to be a machine and technology independent system, robust against slight modifications to the underlying layout model.

Although the goals of the project included implementing a complete system, emphasis was to be on design rather than on implementation. Thus, the overriding objective was to study how one might realistically go about building an automatic layout system, rather than to construct a production-quality software system. The purpose in constructing a prototype system was primarily to give the algorithm designers practical feedback on their algorithms and to ensure that they had considered

all important issues.

Finally, as part of the PI Project, the PI System was to provide a context in which general issues in VLSI layout could be studied.

9.1.2 Input/Output Specifications

Input to PI consists of a set of modules and a set of nets that specify how the modules are to interconnected. PI accepts input expressed in the DPL language [341] or in PI's own simple input language.

Modules are arbitrarily sized rectangles. Each module is designated as either a *pad* or a *logic module*.[1] Each module has a number that indicates its power requirement, and each pad has a preferred orientation. The power and ground pads are identified appropriately. In addition, each module may have connection points, called *pins*, which are represented as line segments on specific layers of the module's border.

Desired interconnections among modules are specified through nets. Each *net* is a set of terminals, and each *terminal* is a set of pins on some module electrically connected within the module. (Multi-terminal nets are allowed. Moreover, a terminal may have multiple pins, and a module may have multiple terminals on the same net.) Each net is labeled as either a *power net*, a *ground net*, or a *signal net*.

Except for the power and ground pads, each module must have at most one power terminal and at most one ground terminal. The user can optionally specify in what order some or all of the pads should be placed along the periphery of the chip.

Output from PI describes a complete, exact, legal layout of the input circuit. PI can express its output in the standard CIF format [341] or in PI's internal representation as a set of rectangular chip features and their locations.

In addition to its formal input, PI also depends on several technology parameters and on several control parameters and switches that guide some of the component algorithms.

9.1.3 Modes of Operation

Although PI is intended to be used primarily for both placement and routing, it is possible to use PI to carry out a subsequence of its layout process. For example, PI can be used to perform module placement, power-ground routing, signal routing, or compaction only.

The ease with which PI can perform an isolated segment of the layout process results largely from PI's problem decomposition. For example, while the placement phase takes routing considerations into account, the routing phase does not depend on any information computed during the placement phase other than the exact positions of the modules.

[1] PI also supports *logo modules* which can be used to put labels, logos, or blank space on the chip.

Since PI was not designed for interactive use, it is not possible to have PI complete a placement or routing only a portion of which is supplied by the user. Many of these interactive features, though, could be added to PI without too much difficulty.

9.1.4 Layout Model

PI lays out chips designed in the Mead/Conway style for *n*MOS and CMOS technologies [341]. This section describes PI's layout model in detail. Constraints of PI's layout model result both from the physical model for chip fabrication and from working assumptions made within PI.

The entire layout must fit within a rectangular region called the *chip*. Modules may be flipped and rotated, but must be aligned with the edges of the chip. All pads must be placed on the chip's periphery. Modules may not overlap. *Chip features* (*e.g.* modules, wires, vias, pins) are not required to fall on any particular grid.

PI assumes that two *layers* (metal and polysilicon) are available for routing. Metal is the preferred routing layer, and all power and ground wires must be routed entirely in metal. In some restricted cases (*e.g.* routing to modules pins in the diffusion layer) PI also allows wires to be laid in the diffusion layer. All wires must satisfy a minimum width rule, and power and ground wires must be wide enough to meet their current-carrying requirements.

Interconnections between modules are made through pins (see section 9.1.2). A routing must electrically connect all terminals on the same net. To connect any two terminals, it suffices to connect any pin on one of the terminals to any pin on the other terminal.

Wire segments consist of rectangles, each of which must be placed parallel to the edges of the chip. Whenever two wire segments touch on the same layer they are electrically connected. Wires on different layers can be electrically connected by means of a *contact cut* (also called a *via*).

Each module is treated as an atomic unit. The dimensions and pin locations of any module may not be modified, and routing is not allowed over any module. However, routing through a module is allowed in the following restricted sense: any two pins on the same terminal are assumed to be electrically connected within their module (by minimum-width wires). Thus, routing through a module is allowed only via fixed, prespecified connection points.

Any two electrically unconnected features must be separated. The required minimum separation depends on the type of features, on what layers the features lie, and on the technology parameters. Electrically unconnected wires on different layers need not be separated, provided they overlap only for short distances as determined by the technology parameters.

9.1.5 Major Design Decisions

This section briefly describes the major decisions that were made in the design of the PI System.

The first major decision, which was more of a problem choice than a design decision, was to build an automatic layout system that would work with a minimum of human interaction. We took as a model an optimizing batch compiler for a high-level programming language. This decision was a natural consequence of our goal to explore how to automate completely the layout phase of VLSI design.

The second major decision was to model the modules as arbitrarily shaped rectangles and to require all chip features to be placed parallel to an edge of the chip. We thought that these working assumptions simplified our task without distorting the essence of the layout problem. Moreover, in practice most modules are rectangular and any nonrectangular module can be represented by a bounding rectangle.

The third major decision was to decompose the layout process as a sequence of subproblems, to be chosen as independent of each other as possible without neglecting important layout considerations. Since the layout problem is NP-complete, we did not attempt to find an optimal layout algorithm. Instead, our strategy was to develop a heuristic layout algorithm by choosing a problem decomposition and by fine-tuning the solutions to each of the subproblems. We reasoned that a sound problem decomposition would help organize the system, control the complexity of the layout process, and focus attention on important restricted layout problems.

The fourth major decision was our particular choice of problem decomposition. Specifically, we decided to separate the placement and routing phases and to decompose the routing problem into independent, fixed-sized switch-box channel routing problems. Fixed-sized channels were needed to keep the channel routing problems independent of each other.

Our fourth major decision went hand-in-hand with two other other important decisions: to work with absolute coordinates rather than symbolic coordinates, and to rely on a resizer to expand congested areas of the chip. We reasoned that absolute coordinates were needed to create independent channel routing problems. We also believed that working with absolute coordinates would help the placement algorithms maintain a more accurate view of what the final placement would look like, thereby achieving better results. The resizer was needed to guarantee that PI would always find a layout.

Finally we made two other decisions, which, at the time, we considered of relatively minor importance. To maintain flexibility, we decided not to require chip features to lie on a global grid. We also decided to represent the layout as lists of rectangular chip features. To conserve space, we decided that wire rectangles would not contain pointers to their neighboring chip features.

9.1.6 Layout Representation

This section briefly describes how PI represents layouts. PI's layout representation affects the time and space complexity of the layout algorithms, as well as the ease of implementing them.

During the different phases of the layout process, PI represents the layout in different ways. The placement algorithms use a data structure called the *placement tree*; the slice and Lee routers use *irregular channel grids*; and the resizer uses a *constraint graph*. But the primary layout representation is a simple data structure known as the *pi-problem*, consisting of lists of rectangular chip features.

The pi-problem represents a layout as a set of rectangular objects organized in a network of lists. The pi-problem consists mainly of a pad list, a logic module list, a net list, a channel list, and a list of channel edges. Each module has a terminal list, and each terminal therein contains a list of pins. Each net has a terminal list together with a list of wire segments. The pi-problem also has numerous other specialized fields used by various component algorithms.

Within the pi-problem, each rectangle is represented by its lower-left and upper-right corner points, expressed in a global coordinate system. All objects except wire segments have names. Moreover, modules, channels, and channel edges contain pointers to their neighboring modules, channels, and channel edges, if any. Each wire rectangle has a layer assignment, but no pointer to any adjacent chip feature. Each via is denoted only by a wire rectangle with a special layer assignment.

9.2 Definitions and Notations

This section defines terms and notations used throughout part II of this dissertation.

Circuit Graph Terms

The following straight-forward definitions provide a convenient vocabulary for reasoning about PI's modules and nets as a graph.

Let the *circuit graph* denote the multi-hypergraph that describes PI's input. The nodes of this graph are the modules; the hyperedges are the nets. The circuit graph is a hypergraph because nets may have more than two terminals. The circuit graph is a multi-hypergraph because different nets may connect the same set of modules. Although it is convenient to view PI's input as a multi-hypergraph, this analogy is slightly imperfect since any module may have two distinct terminals on the same net.

For any module M, let *terminal-list*(M) denote the list of terminals on M. Similarly, for any net N, let *terminal-list*(N) be the list of terminals on N. Whenever N has a terminal on M, we say that N *touches* M.

For any terminal t, let *module*(t) and *net*(t) denote respectively the module and net associated with t.

Let N be any net and let M be any module. The *length of* N is the the number of terminals on N; the *degree of* M is the number of terminals on M. Should it be necessary to distinguish between the number of terminals on N and the number of modules on N, we will use the following more specific terminology. The expression *t-length*(N), read "*terminal length of* N," denotes the number of terminals on N, and the expression *m-length*(N), read "*module length of* N," denotes the number of modules on N. Similarly, the expression *t-degree*(M) denotes the number of terminals on M, and the expression *n-degree*(M) denotes the number of nets on M.

For any module M, let *area*(M) be the area of M, *i.e.* the area of the rectangle defining M. More generally, for any set of modules $\mathcal{M}$, let *module-area*($\mathcal{M}$) be the sum of the areas of the modules in $\mathcal{M}$.

Layout Terms

Design rules for chip fabrication include numerous minimum-separation and minimum-width requirements. Let *min-wire-width* and *min-wire-sep* denote respectively the minimum wire width and minimum wire separation imposed by the design rules.[2] Finally, let *min-track-width* be the sum of *min-wire-width* and *min-wire-sep*.

The following terms refer to special situations that arise in wire routing. A *bend* is a (ninety degree) turn in a wire on the same layer. A *via* (or *contact cut*) is where a wire changes layers, and a *crossover* occurs when two perpendicular wires cross over each other separate layers.

Three important measures of a layout are *chip area*, *wire length*, and *wire area*. Chip area, or simply *area*, is the area of the smallest rectangle that contains the entire layout. Wire length is a linear measure of the amount of wire in the layout, and wire area is a two-dimensional measure of the amount of wire in the layout.

Geometric Terms

The following geometric terms are helpful to describe PI's placement algorithms.

The *aspect ratio* of a rectangle is the ratio of the length of the rectangle's shortest side to the length of its longest side.

A *bounding box of an object (or objects)* is a rectangle that contains the object (or objects). The *minimum bounding box of an object (or objects)* is the smallest area bounding box of the object (or objects). We shall sometimes refer to the minimum bounding box of an object (or objects) as the *extent* of the object (or objects).

[2] For simplicity, we will ignore the fact that PI's layout model allows separate minimum widths and separations for each layer. The reader may interpret the design rule terms defined in this section as the maximum minimum-widths and minimum-separations taken over all layers.

A *partition of a rectangle* is a division of the rectangle into nonoverlapping regions that cover the rectangle. A *slicing of a rectangle* is a special type of partition, recursively defined as follows. A slicing is either a rectangle, or it is a partition of a rectangle into precisely two slicings. Thus, any slicing can be formed by recursively cutting a rectangle into two rectangular regions.

An *orthogonal transformation* is either a flip (a reflection) or a rotation. A *rigid transformation* is the composition of an orthogonal transformation with a translation. The *orientation of a module* is an orthogonal transformation around the center of the module.

Unless otherwise specified, distances in the PI System are measured by the so-called *Manhattan metric*. If P_1 and P_2 are points in the plane with Cartesian coordinates (x_1, y_1) and (x_2, y_2) respectively, then the Manhattan distance between P_1 and P_2 is given by

$$dist(P_1, P_2) = |x_2 - x_1| + |y_2 - y_1| \tag{9.1}$$

where $|x_2 - x_1|$ denotes the absolute value of $x_2 - x_1$.

Distances between pins and terminals are defined as follows. The distance between any two pins is the Manhattan distance between the pin centers. The distance between any two terminals is the minimum distance between any pair of pins from the two terminals.

Chapter 10

The PI System's Post-Placement Algorithms

To understand how PI's placement algorithms interact with the rest of the PI System, it is important to know more about PI's routing and resizing algorithms. After placing the pads and logic modules, PI lays the wires in separate power-ground routing and signal routing phases. During routing, PI attempts to minimize wire length, number of bends, and number of crossovers. Module positions remain fixed throughout routing. However, if routing fails, then PI calls the resizer to expand the chip and routing is reattempted. This chapter gives a brief overview of PI's routing and resizing algorithms.

10.1 Power-Ground Routing

Because the requirements for power-ground routing are much more restrictive than those for signal routing, PI routes the power and ground wires before routing the signal wires. To route the power and ground wires, PI first supplies power and ground to the pads and then to the logic modules. If PI is unable to route all of the power and ground wires, then the resizer is called to enlarge the chip and power-ground routing is reattempted.

PI lays the power and ground wires using a special-purpose routing strategy specifically designed to handle the special requirements of power-ground routing [446]. These requirements differ from those for signal routing in three important respects. First, since each module requires power and ground, power and ground wires must span the entire chip. Second, all power and ground wires must be laid entirely in metal. This second requirement is especially confining in the one-metal-layer model used by PI, for it implies that power and ground wires must not cross each other. Third, power and ground wires must be wide enough to meet their current requirements. Subject to these requirements, PI attempts to find a power-ground routing

that minimizes the amount of wire used.

PI supplies power and ground to the pads through two broken rings. PI lays a ground ring between the pads and the edges of the chip, and a power ring between the pads and the logic modules. To enable ground to be brought into the chip's center, PI leaves a gap in the power ring.

After supplying power and ground to the pads, PI is faced with the problem of supplying power and ground to the logic modules, where the logic modules are surrounded by a power ring. PI proceeds in two steps. First, PI grows a ground tree to provide the modules with ground. PI gives the modules ground before power since the power ring makes power routing more flexible than ground routing. Second, PI extends power wires from the power ring to the logic modules. Each of these two steps is carried out with the help of the signal routing routines.

Since each module has at most one power terminal and at most one ground terminal, when power and ground wires are routed in the same layer, it is always topologically possible to route the power and ground nets as interdigitated trees. It can happen, however, that one of the trees will wrap around the other, thereby forcing the inner tree to take a circuitous route. While this problem seems to be less severe when a power ring surrounds the logic modules, it can nevertheless happen.

To help prevent the ground tree from interfering with the power routing, PI has a novel heuristic that keeps the ground tree within a central region of the chip. PI defines this central region by a short "traveling salesman tour" (Hamiltonian circuit) of the logic modules, calculated by Lin's "3-swap" approximation algorithm [364]. Since this region separates a power and ground pin on each module, it is always topologically feasible to route the power and ground wires with this heuristic.

Although the Hamiltonian circuit heuristic has been implemented, PI does not normally use this heuristic. Instead, PI routes the ground tree with the signal routing algorithms, adapted for this purpose. The primary reason for excluding this heuristic is that it is rather time consuming—Lin's approximation algorithm runs in time $O(m^2)$, where m is the number of modules.

Finally, PI computes the current requirements for each segment of the power-ground wires and then uses the resizer to stretch the wires to their desired widths.

10.2 Signal Routing

PI routes the signal wires in four steps: channel definition, global routing, crossing placement, and channel routing. *Channel definition* partitions the chip into rectangular regions, thereby establishing a structure in which the routing problem can be decomposed into local routing subproblems. *Global routing*—also referred to as *coarse* or *loose* routing—determines the coarse path of the wires through channels (and thus around modules). PI's novel *crossing placement* step fixes all intercon-

nections between channels, thereby decoupling the remaining routing task into independent fixed-sized switch-box channel routing problems. Finally, *channel routing* performs detailed local routing within each channel. If either the global routing or channel routing routines are unable able to complete a routing, then PI calls the resizer to expand the chip and signal routing is reattempted.

10.2.1 Channel Definition

During channel definition, PI partitions the regions of the chip not occupied by modules into nonoverlapping rectangular *channels*. Each channel is a *switch-box* in the sense that it has fixed height and fixed width and may have pins touching any or all sides of the channel. PI defines large, "natural-looking" channels by using a heuristic that attempts to minimize the sum of the lengths of the channel edges. As shown by Lingas, Pinter, Rivest, and Shamir, minimizing the sum of the lengths of channel edges is NP-complete [465].

PI identifies two exceptional channel types for special treatment—*narrow channels* and *covered channels*. Narrow channels are channels that are so small that wires cannot change direction within them; covered channels are the regions under power and ground wires. No crossovers are possible within covered channels.

10.2.2 Global Routing

During global routing, PI determines the set of channels through which each net will travel. PI routes the nets one by one, ordering them by increasing extent and decreasing number of pins. For each net, PI applies a minimum-cost Steiner tree heuristic [443,449,347,349].

PI casts the global routing problem as a graph problem involving an undirected *channel graph*. The vertices of the channel graph are the module pins of the current net and the channel-edge midpoints. Two vertices are joined by an edge if and only if they lie on the same channel. For each net, PI attempts to find a minimum-cost Steiner tree in the channel graph that spans all terminals on the net. Cost is taken to be Manhattan distance, with penalties for traveling around turns, across covered channels, and through congested areas.

For a two-pin net, PI uses Dijkstra's shortest-path algorithm [18]. For a multi-pin net, PI uses a more complicated Steiner-tree heuristic in which the net is routed in a series of iterations. During each iteration, simultaneous searches expand from each vertex along the partial routing of the net in the channel graph. Each iteration terminates when any search meets a pin not yet connected. Only the successful search path is added to the partial routing.

Output from the global channel router is represented by two data structures—*crossings* and *strands*. A crossing represents a net crossing a channel edge, and a

strand represents a piece of a net that traverses a particular channel. Each edge has a list of crossings, and each channel has a list of strands. It is possible, though unlikely, for a channel to have more than one strand for the same net.

10.2.3 Crossing Placement

After global routing, only a coarse routing of the wires has been determined: for each net, PI knows only through what channels the net will pass. Crossing placement splits the detailed routing problem into independent switch-box channel routing subproblems by determining the exact positions and layers where nets cross edges that separate channels. This novel step attempts to minimize globally the required wire length and the number of forced crossovers, thus separating global aspects of detailed routing from local concerns.

Without a crossing placement step, the way in which nets cross channel edges would be left up to the channel routers. This would imply that the channel-routing subproblems would depend on each other, and that the order in which channels are routed would be important. Moreover, to route any local channel wisely, the channel router would have to keep global considerations in mind.

Currently, PI uses an iterative relaxation heuristic. Section 13.2 describes a new crossing placement algorithm that was never implemented [448].

The iterative relaxation heuristic proceeds edge by edge. Initially each crossing is temporarily placed at the center of its edge. For each edge, the position of each crossing on the edge is adjusted. To adjust the position of a crossing on its edge, PI computes a weighted average of the old crossing position and the projections of the other crossings on the same strand onto the edge. The effect of this procedure is to shorten strands and to align crossings on the same strand. Except for crossings that are module pins or that are aligned directly across from module pins, PI positions all crossings on a global grid. After PI determines the positions of all crossings, a straight-forward layer assignment step completes the crossing placement heuristic.

10.2.4 Channel Routing

After crossing placement, PI solves the resulting independent, switch-box channel-routing problems. PI applies an array of different channel routers in turn, from fastest to slowest, until one succeeds. If all three channel routers fail, then PI calls the resizer to expand the chip and routing is reattempted. PI uses three routers: a simple and fast pattern router called the *quick router*, a left-to-right scanning router called the *slice router*, and "Lee-style" shortest-path router [427] called the *Lee router*.

Each channel is a fixed-sized rectangle with strands extending to and from any combination of sides.[1] Strands enter and leave the channel at fixed positions that cor-

[1] A *strand* represents a connected piece of a net within a channel. A channel might have two

respond to pins and crossings. PI finds a detailed routing of each strand, attempting to use a minimum amount of wire, jogs, and crossings. All routing must be contained within the channel and the channel routers must take care not to create design rule violations by running wires too close to objects in neighboring regions.

- The quick router has a small library of single-strand routing patterns. For each strand, the router simply checks if the net can be routed by any of the patterns. Each strand is routed completely independently of the other strands, possibly creating design rule violations. As a final step, if each strand is routed, a design rule checker determines if the complete channel routing is legal.

- PI's slice router adapts the "greedy-router" methodology described by Rivest and Fiduccia [450] to the switch-box context [444]. The slice router first identifies either the horizontal or vertical dimension of the channel as the primary dimension. Starting at one side of the primary dimension, the router proceeds slice-by-slice (*i.e.* column-by-column or row-by-row) until it reaches the opposite side. Each slice is routed before proceeding to the next slice, and the routing is guided by a collection of greedy heuristics.

- PI's Lee router sets up a grid and then routes the strands within the grid one strand at a time. The Lee router routes each strand in essentially the same way in which the global router routes each net. The cost function used to guide the search for a good routing of each strand depends on several adjustable parameters which control a variety of heuristics.

10.3 Resizing

The *resizer* is a program that expands or shrinks a layout while preserving certain invariants, such as not changing the size of modules. PI uses its resizer for a variety of purposes, including widening the power and ground wires to meet their current requirements, enlarging congested regions of the chip that could not be routed, and compacting the layout. The resizer plays a crucial role in guaranteeing that PI will always find a layout.

The resizer is flexible in the amount of wiring that it preserves: it can be instructed to eliminate all, none, or some of the wiring. For example, when used to expand the chip, the resizer might be instructed to leave all power and ground wires, but no signal wires.

Except for a special *uniform dilation mode* in which the resizer expands all free areas of the chip by a constant factor in length, channel structure can change as a

---

different strands corresponding to the same net.

result of resizing. Consequently, the entire signal routing phase, including channel definition, is usually recomputed whenever any channel fails to route successfully.

When called as a result of routing failures, the resizer can accept detailed reports from the routers about the routing failures. For example, the global router can identify overstuffed channel edges.

PI's resizer is bi-directional in the sense that it operates in successive horizontal and vertical passes. During each pass, a *constraint graph* is constructed and solved. Each node of the constraint graph denotes a side of some chip feature, such as a module, a wire rectangle, or the chip. Each edge represents a constraint between two features and is specified by an inequality of the form $x_1 \leq x_2 + c$, where x_1 and x_2 are the coordinates of the features and c is a rational number. Recognized in part through a scanning process, constraints can result from design rules, current requirements, layout topology, routing considerations, and feature sizes. PI solves the constraint graph using Gaussian elimination. In carrying out Gaussian elimination, PI uses a standard heuristic in which the nodes to be eliminated are ordered by decreasing product of in- and out-degree.

The resizer alone does not guarantee that PI will eventually find a layout: PI's convergence property also depend on how channel definition, global routing, crossing placement, and channel routing perform. To guarantee that PI will eventually find a layout, the following conditions must be true. Of course, each component algorithm must eventually terminate and produce legal output. More interestingly, channel definition must not produce too small channels. Global routing must eventually succeed on some expansion of the placement, and global routing must not unnecessarily use excessive amounts of wire (say, by maliciously winding wires around modules). Crossing placement must not crowd crossings into channel corners. The resizer must expand the layout sufficiently, and detailed routing must eventually succeed on some expansion of the layout.

Chapter 11

The PI System's Placement Algorithms

This chapter gives a high level description of the PI System's placement algorithms, concentrating on the broad structure of the algorithms, PI's placement problem, and the framework in which the algorithms operate.

11.1 Overview of PI's Placement Algorithms

PI places the modules in three major steps. First, PI estimates the size and shape of the chip. Second, PI places the pads around the periphery of the chip. Third, PI places the logic modules in the central region of the chip known as the *logic box*. These steps are called *estimation of chip size and shape*, *pad placement*, and *logic placement* respectively.

The emphasis of PI's placement algorithms is on the logic placement step, which is further decomposed into the problems of computing a placement hierarchy, orienting the modules, and leaving space for routing. PI builds the placement hierarchy using a top-down recursive mincut process that recursively partitions the circuit graph while simultaneously slicing the logic box into rectangular regions. The placement hierarchy is a type of approximate placement in which the approximate location of each module is known. To transform the placement hierarchy into an exact placement, PI first determines how each module should be flipped and rotated. Then, PI successively glues modules and supermodules together in a postorder traversal of the placement hierarchy. Known as *hardening*, this final bottom-up process leaves space for routing and aligns the modules to facilitate routing.

Throughout the placement process, PI represents the layout using a data structure called the *placement tree*. This data structure maintains a variety of graph-theoretic and geometric information about the layout in a way that can be easily manipulated. The placement process can be viewed as a set of operations that successively refine

1. Initialize placement tree
2. Estimate size and shape of logic box
3. Place pads around chip's periphery
4. Place logic modules
   (a) Find an approximate placement of the logic modules by computing a complete breadth-first mincut decomposition of the placement tree
   (b) Orient each module
   (c) Calculate an exact placement of the logic modules by successively gluing the modules and super modules together in a postorder traversal of the placement tree
5. Adjust pads
6. Flatten placement tree

Figure 11.1: Outline of PI's placement process

an initial placement tree through a series of approximate placements into an exact final placement.

Figure 11.1 summarizes PI's placement process. For completeness, this figure includes three minor placement steps not yet described. The first additional step is the initial step, which constructs an initial placement tree. The two other additional steps follow logic placement. In the penultimate step of the placement process, PI adjusts the locations of the pads to accommodate any change in the estimated logic box size that may have occurred during hardening. In the last step of the placement process, PI "flattens" the placement tree by expressing the location of each module in a common coordinate system. This last step is necessary because the placement tree uses a hierarchy of different coordinate systems.

11.2 PI's Placement Problem

During its placement phase, the PI System determines a nonoverlapping placement of the modules in a rectangular region. PI attempts to leave enough room for routing, while minimizing the amount of resources required to route the placement. Thus, PI's placement problem is an optimization problem with routing constraints. PI's approach to placement differs from that of symbolic layout systems, such as the

Phoenix system [397], both in that PI attempts to leave enough room for routing and that PI computes an absolute position for each module that will remain fixed during routing.

Input to PI's placement problem is a set of arbitrarily sized rectangular modules and a set of nets which describe how the modules are to be interconnected. Modules may be flipped and rotated, but must be placed parallel to the edges of the chip. Pads must be placed around the chip's periphery.

The main resource that PI's placement algorithms attempt to minimize is total estimated layout area, which includes space left for routing. PI also attempts to minimize estimated wire area, but when unavoidable tradeoffs between chip area and wire area arise, PI gives preference to minimizing chip area. At least to some degree, through PI's concern of layout area and wire area and through PI's treatment of module alignments, PI's placement algorithms also implicitly attempt to avoid forcing unnecessary bends and crossovers.

The amount of space left between modules is determined primarily by routing area estimates. While these estimates work well in practice, they are not guaranteed to leave enough space for routing. Thus, it is possible for PI to produce unroutable placements. If this happens, the condition is detected during routing, at which time PI's resizer is invoked to expand the layout.

11.3 The Placement Tree

The *placement tree* is a data structure that represents the placement of the modules throughout PI's placement phase. By maintaining hierarchical and geometrical layout information, the placement tree can represent several types of approximate and partial placements. This data structure supports a set of operations, called *refinements*, which PI uses to transform an initial approximate placement into a finished placement.

The placement tree is a tree, each subtree of which represents an approximate placement of some of the modules. Each node of the placement tree is an object called a *pi-box*. The root pi-box corresponds to the chip, and each leaf pi-box corresponds to a module. For each module, there is precisely one associated pi-box. Each node points to its parent and children. Initially, the placement tree consists only of the root pi-box and its children, the module boxes.

To place the modules, PI applies a sequence of refinements to the placement tree. These refinements modify the structure of the placement tree as well as the state associated with the pi-boxes. The rest of this section explains the placement tree in greater detail and discusses the various types of approximate placements that it can represent.

Placement Tree Terminology

It is convenient to have names for certain special pi-boxes. The *chip box* is the root of the placement tree; the *logic box* is the pi-box that is associated with the placement of all logic modules; and a *module box* is a pi-box that is associated with a module.

A *unary pi-box* is any pi-box that has exactly one child. A *binary pi-box* is any pi-box that has exactly two children. Any pi-box that is not a leaf of the placement tree is called an *internal pi-box.*

The *descendant modules of a pi-box* are the modules corresponding to the leaves of the placement subtree rooted at the pi-box. For any pi-box P, the placement subtree rooted at P represents an approximate placement of P's descendant modules. This approximate placement is called the *partial placement represented by* P, or simply P's *partial placement.* The placement represented by P is *partial* in the sense that it might involve only a subset of all modules to be laid out.

The placement subtree rooted at the logic box is called the *logic tree.*

We say that a *net touches a pi-box* if and only if the net touches some descendant module of the pi-box.

Pi-Box Properties

In addition to its parent and children, each pi-box has five other important properties—transform, extent, virtual pin list, module area, and status. These properties constitute the *state* of the pi-box and give information about the approximate placement represented by the pi-box. As used by PI, the virtual pin list and module area fields are included for efficiency and convenience only and do not add any additional information to the placement tree that could not be computed from other fields.

To support geometric operations on the placement tree, each pi-box has its own coordinate system. The *transform* of a pi-box describes how the pi-box's coordinate system relates to its parent's coordinate system. Each transform must be a *rigid* transformation (*i.e.* a translation followed by flip or rotation).

The *extent* of a pi-box is a rectangular region intended as a bounding box for the placement of the pi-box's descendant modules. Each extent is represented by lower-left and upper-right corner points, which are specified in the coordinate system of the pi-box.

For each pi-box, for each net touching the pi-box, there is an associated *subnet* that consists of all terminals on the pi-box's descendant modules.[1] For each such subnet, the pi-box maintains a *virtual pin (vpin)*, which specifies the minimum bounding box containing all pins on the subnet. Vpins are useful for keeping track of the approximate regions in which subnets will be routed. Each vpin consists of a net

[1] A *subnet of a net* is a subset of terminals on the net.

and a bounding box. The bounding box of each vpin is represented by lower-left and upper-right corner points, which are specified in the coordinate system of the pi-box. The *virtual pin list (vpin list)* of a pi-box is a list of all vpins associated with the pi-box.

The *module area* of a pi-box is the sum of the areas of the pi-box's descendant modules.

The *status* of a pi-box describes the pi-box's stage of refinement. As explained in the next subsection, for the refinements used by PI, a pi-box can be either free, planned, or hardened.

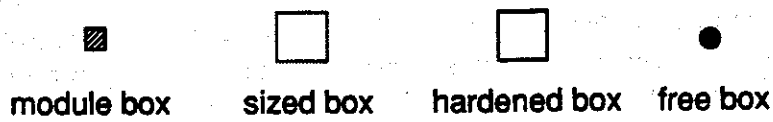
Types of Approximate Placements

During the placement process, the placement tree undergoes various stages of refinement. These stages of refinement correspond to different types of approximate placements which can be described in terms of the state of the pi-boxes.

A pi-box is *oriented* if its transform has been fixed. For example, an oriented module box represents a module whose orientation is known. A pi-box is *sized* if it has an extent. Sized pi-boxes correspond to approximate placements whose bounding area is known or estimated. In addition, a pi-box can be either free, planned, or hardened. A *planned pi-box* has a "floorplan" which specifies where in the pi-box's extent the children will lie. A planned binary pi-box arises naturally after each mincut step and represents an approximate placement consisting of two parts; each part is to be laid out in a rectangle and the rectangles are to be placed next to each other. A *hardened pi-box* represents an exact legal placement. A *free pi-box* is neither planned nor hardened.

A variety of types of partial placements is illustrated in figure 11.3, which shows an approximate placement and its representation as a placement tree. In this figure, the exact placements of modules A, B, and C have been determined. Although their exact locations are not yet known, modules D, E, F, and G will be placed somewhere beneath modules B and C, and module G will be placed to the right of modules D, E, and F. It has not yet been determined how module G will be flipped and rotated. The reader may find it helpful to refer to figure 11.2, which summarizes the conventions used to draw all placement tree diagrams in part II.

A planned pi-box is a sized binary pi-box whose extent is partitioned into two regions, with each region corresponding to exactly one child. Thus, a planned pi-box has a "floorplan" which specifies where in the pi-box's extent the children will lie. Children of planned pi-boxes must be sized. Moreover, each child's extent must have the same dimensions as the corresponding region of the floorplan, and each child must be translated to lie in its "room" of the floorplan. Planned pi-boxes, however, do not necessarily represent legal placements since a grandchild of a planned pi-box might not fit into the pi-box's extent.



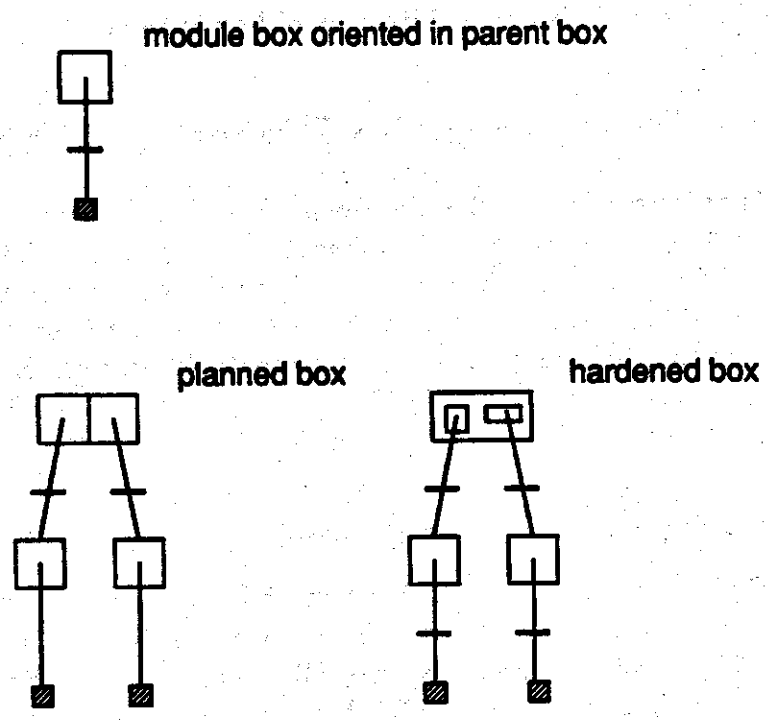


Figure 11.2: Conventions for drawing placement trees. In all figures of part II, pi-boxes are denoted by dots or rectangles. A dot represents an unsized pi-box; a rectangle represents a sized pi-box. A darkened square stands for a module box. Rectangles drawn with solid lines denote hardened pi-boxes, and rectangles drawn with dashed lines denote free or planned pi-boxes. For each planned or hardened pi-box, the positions of the children are drawn within the square that represents the pi-box. A hatch mark on the parent edge of any pi-box indicates the pi-box has been oriented.

Although the partition of a planned pi-box is represented through the transforms of the children, the pi-box also contains a field that identifies the *partition orientation.* A *horizontal partition* results when the extent is partitioned by a line parallel to the x-axis of the chip; a *vertical partition* results when the extent is partitioned by a line parallel to the y-axis.

A hardened pi-box is a sized pi-box that satisfies the following three properties: The children must be hardened; the children's extents must lie within their parent's extent; and the children's extents must not overlap. The children's extents, however, are not required to partition their parent's extent.

A *placement subtree is planned* if and only if every node of the subtree is planned. The *subtree is hardened* if and only if its root pi-box is hardened.

11.4 How PI Refines the Placement Tree

A *refinement* is an operation on the placement tree that attempts to improve or to make more specific the approximate placement represented by the placement tree. Refinements work by modifying the structure of the placement tree and by modifying the properties of the pi-boxes. PI places modules by successively refining an initial placement tree until reaching a finished placement. This section summarizes PI's refinement process by describing the initial placement tree, the refinements used by PI, and the effects these refinements have on the placement tree.

11.4.1 The Initial Placement Tree

The initial placement tree consists of a chip box whose children are module boxes. For each module, there is exactly one module box. Each pi-box transform is the identity transformation (Typically, this means that all modules will initially overlap with their lower left corners at the origin). The chip box is free and unsized. Each module box is sized and hardened to the exact dimensions of the corresponding module (see figure 11.4).

11.4.2 The Refinement Process

This section summarizes PI's placement process in terms of its effect on the placement tree.

After constructing the initial placement tree, PI estimates chip size and places the pads around the chip's periphery. PI separates the pads from the logic modules and adds a logic box to the placement tree. The logic modules become children of the logic box, which is a child of the chip box. To estimate chip size, PI sizes the logic box. During pad placement, PI distributes the pads along the edges of the chip.

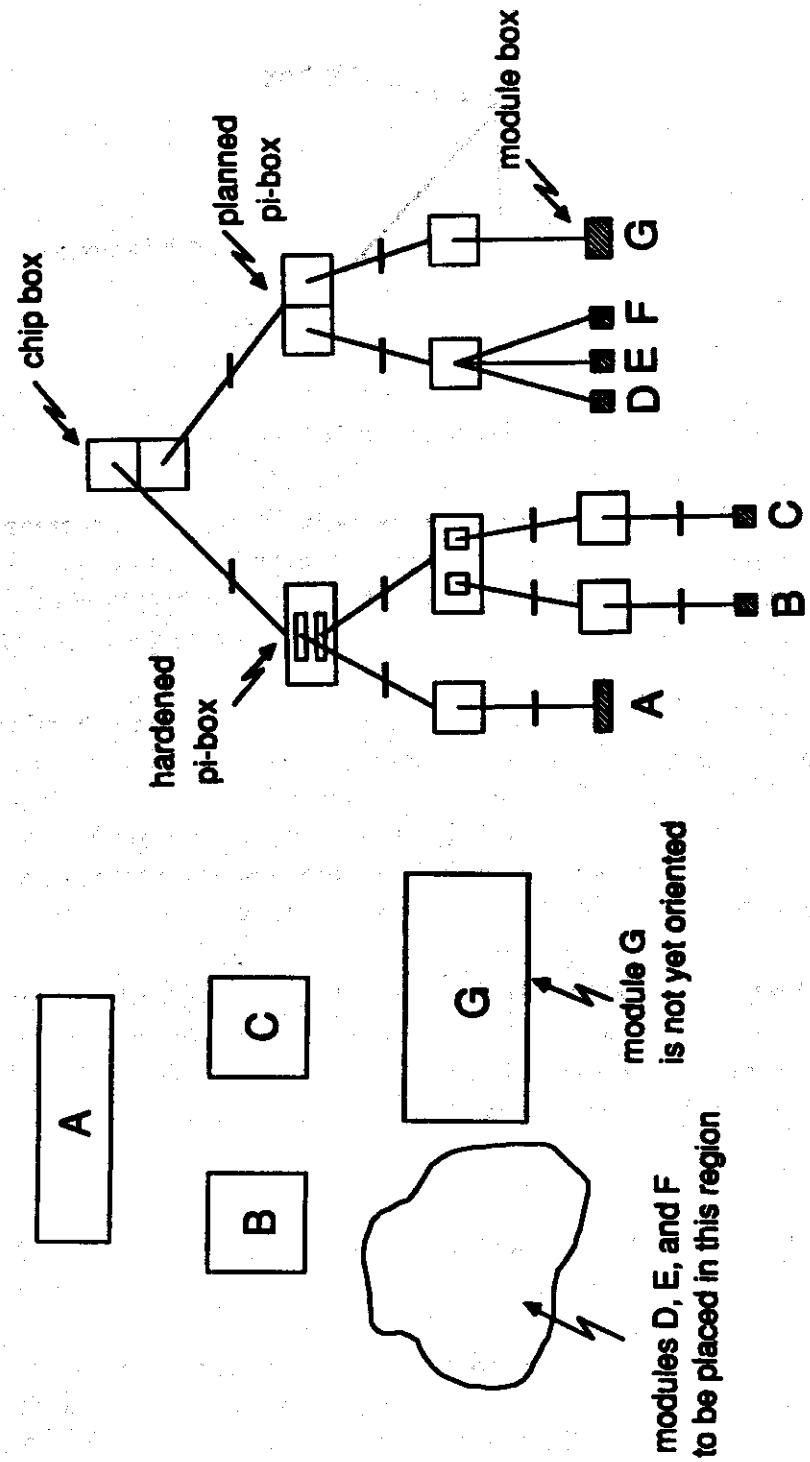


Figure 11.3: An approximate placement and its corresponding placement tree

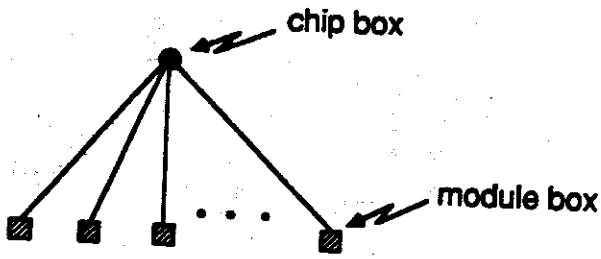


Figure 11.4: Initial placement tree

For each side of the chip that receives one or more pads, PI creates a corresponding *side box* to contain the pads assigned to that side. Each pad box is oriented within its parent side box. Finally, PI hardens and orients each side box within the chip box. Figure 11.5 illustrates the effect of chip size estimation and pad placement on the placement tree.

After laying down the pads, PI places the logic modules in the following three steps. First, PI computes an approximate placement by recursively applying a mincut refinement to the logic tree. This mincut process refines the logic tree into a planned, binary tree (see figure 11.6). Second, PI orients the module boxes. Third, PI determines an exact placement by recursively hardening the logic tree in postorder. This hardening process PI adjusts the translation and extent of each internal pi-box in the logic tree.

After placing the logic modules, PI adjusts the pads to accommodate any change in the extent of the logic box that may have occurred during logic placement. To adjust the pads, PI translates the side boxes appropriately. Finally, PI hardens the chip box and computes the coordinates of each module in the chip's coordinate system.

11.4.3 PI's Mincut, Orientation, and Hardening Refinements

PI's placement algorithms are built around the following three operations on the placement tree: *top-down mincut partitioning*, *module orientation*, and *bottom-up hardening*. Input to each of these refinements consists of a pair $(P, \mathcal{T})$, where $\mathcal{T}$ is the placement tree and P is a pi-box in $\mathcal{T}$. Given any such pair $(P, \mathcal{T})$, each of PI's refinements operates on P's partial placement; the placement tree $\mathcal{T}$ provides the

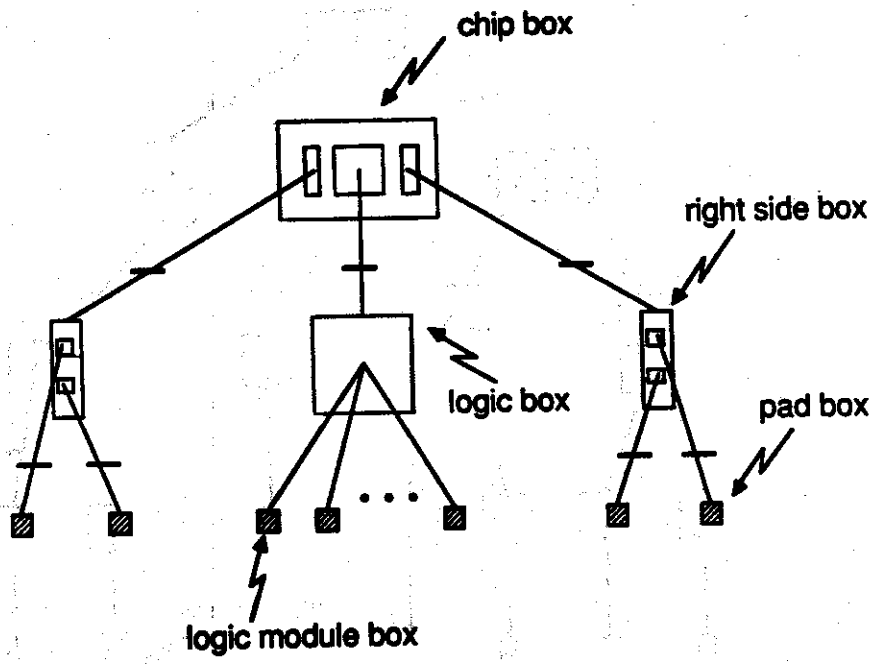


Figure 11.5: Placement tree after pad placement

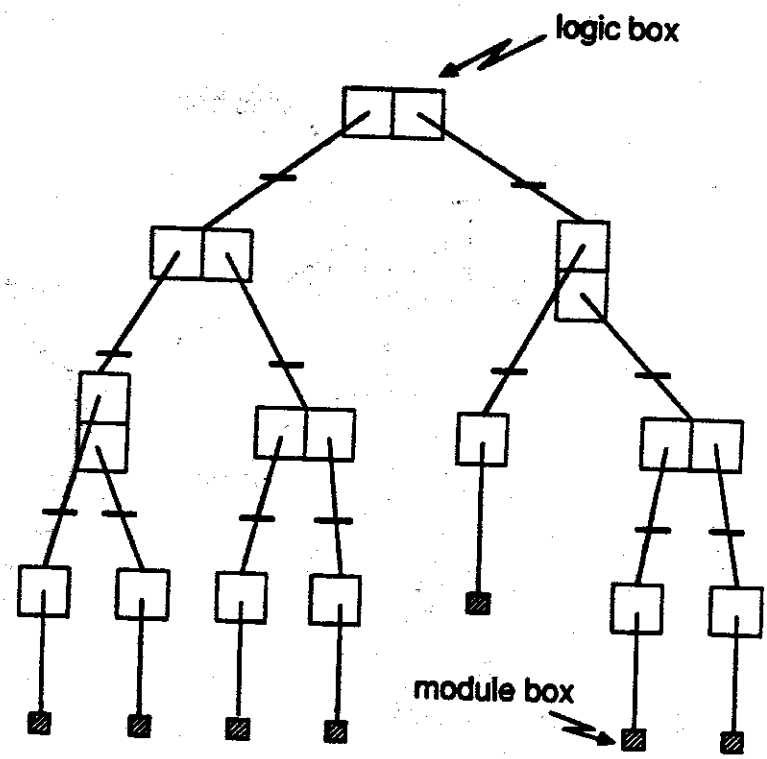


Figure 11.6: Complete mincut decomposition of the logic tree

context in which PI performs local operations on P. This section explains how the mincut, orientation, and hardening refinements affect the placement tree.

Top-Down Mincut

The mincut refinement is a top-down heuristic that refines an approximate placement by partitioning the set of modules in the approximate placement into two subsets and by allocating a tentative region within the approximate placement where each of the subsets of modules will be placed. In partitioning the set of the modules, PI brings together modules that are highly interconnected, taking into consideration the context in which the partition is performed.

Input to the *mincut refinement* is a pair $(P, \mathcal{T})$ such that P is a free and sized pi-box with at least two children. Although the children of P need not be module boxes, they must be sized. The mincut refinement first partitions the children of P into two subsets $\mathcal{A}$ and $\mathcal{B}$. Next, PI creates two new pi-boxes A and B, which become the new children of P. The pi-boxes in $\mathcal{A}$ become the children of A, and the pi-boxes in $\mathcal{B}$ become the children of B. During this refinement, PI also determines a *floorplan* for P. The floorplan is a partition of P's extent into two rectangular "rooms" corresponding to A and B respectively. PI sizes and translates A and B to lie exactly in their respective rooms of the floorplan. Finally, PI sets the partition orientation field of P and declares P planned. At the end of the mincut step, A and B are free, sized, and oriented (see figure 11.7).

Module Orientation

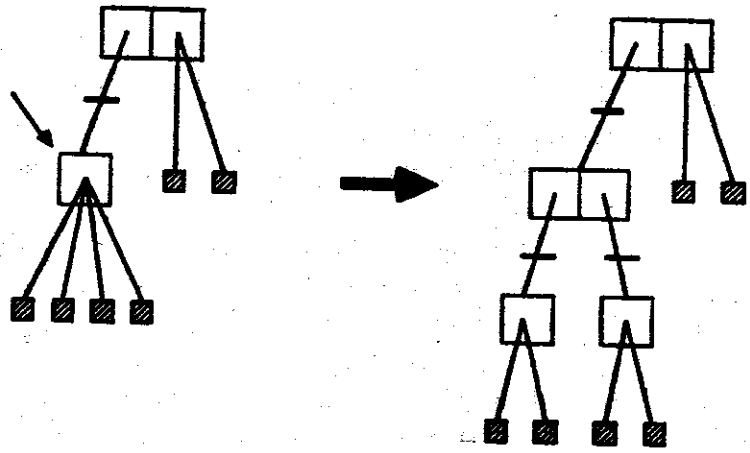
The module orientation refinement improves an approximate placement by flipping and rotating a single module. This heuristic attempts to orient a module to reduce estimated chip area and wire length.

Input to the orientation refinement is a pair $(P, \mathcal{T})$ such that P is a sized unary box whose only child is hardened. This refinement modifies the transform of P's child and declares P hardened; the structure of $\mathcal{T}$ remains unchanged (see figure 11.8).

Bottom-Up Hardening

The hardening refinement is a bottom-up heuristic that determines exactly how two modules (or hardened pi-boxes) should be placed relative to each other. This refinement allocates space for routing between the modules (or pi-boxes) and aligns the modules to facilitate routing.

Given a planned binary pi-box P whose children are hardened, the hardening refinement makes P's approximate partial placement exact. The hardener determines how P's children should be offset and separated from each other, while preserving the orientation of P's children and the orientation of P's floorplan. After translating P's



Note

In this and other placement figures, an arrow identifies the pi-box being refined.

Figure 11.7: Mincut refinement

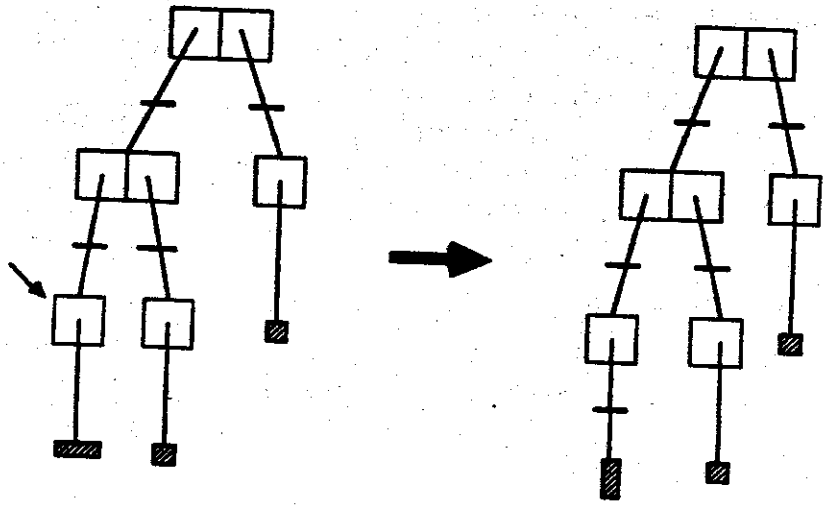


Figure 11.8: Orientation refinement

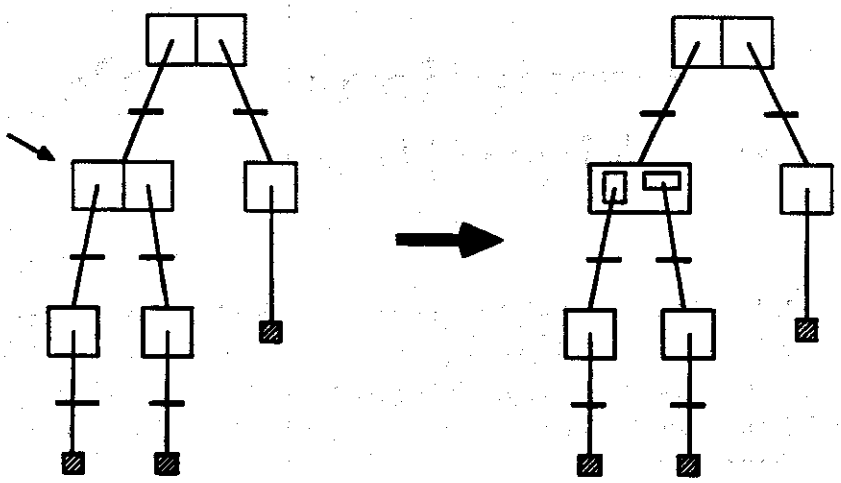


Figure 11.9: Hardening refinement

children appropriately, the hardener expands or shrinks the extent of P to fit tightly around the children. Finally, PI declares P hardened. After hardening, P's children still lie within P's extent, but the children no longer necessarily form a partition. Unlike mincut, hardening guarantees to produce a legal partial placement and does not modify the structure of $\mathcal{T}$ (see figure 11.9).

Chapter 12

Detailed Descriptions of PI's Placement Algorithms

To place the modules, PI estimates the size and shape of the chip, places the pads, computes a mincut placement hierarchy, orients the modules, and finds an exact placement of the placement hierarchy. This chapter describes these steps in detail.

See section 9.2 and chapter 11 for explanations of notation and terminology used throughout this chapter.

12.1 Estimating Chip Size and Shape

To begin the placement process, PI estimates the size and shape of the logic box. Except for certain unusual cases, PI makes the initial logic box a square. PI estimates logic box area taking logic module area and estimated routing area into consideration.

PI's initial estimate of logic box size and shape affects all subsequent placement steps. The more accurate the estimated logic box size and shape are, the more realistic the approximate placements computed during the placement process will be. For example, if estimated chip size grossly underestimates actual chip size, then the mincut refinement tends to produce approximate placements in which many modules overlap. Since the relative positions of pins in overlapping modules can change significantly as a result of hardening, the mincut and orientation steps that depend on such information will perform less well. On the other hand, if estimated chip size grossly overestimates chip size, then the mincut and orientation steps tend to pay less attention to geometric considerations.

12.1.1 How PI Estimates Logic Box Shape

PI chooses the shape of the logic box to be a rectangle of area A, where A is the estimated area of the logic box. To ensure that the longest module can fit into the

logic box, PI takes one side of this rectangle to be the maximum of $\sqrt{A}$ and L_{max}, where L_{max} is the length of the longest logic module. Thus, PI chooses the logic box to be a square, unless a square of area A will not contain the longest module.

The choice of a square logic box was based on the following two considerations. First, for any fixed area, a square is the rectangle with the shortest perimeter. Second, for any fixed area, a square is the rectangle that minimizes the maximum distance between any two interior points. These two facts suggest that a square logic box helps reduce wire length.

If there are more pads than will fit around the logic box, PI simply expands the logic box to accommodate the pads. Currently, PI detects and handles this exceptional case during pad placement. A more careful initial estimation of logic box shape would also take pads into consideration.

12.1.2 How PI Estimates Logic Box Area

PI estimates layout area as the sum of three components: module area, local routing area around modules, and global routing area of nets. Specially, given a set of modules $\mathcal{M}$ connected by n nets, PI computes the sum

$$\begin{aligned} \textit{est-layout-area}(\mathcal{M}) \;=\; & c_1 \cdot \textit{area}(\mathcal{M}) + c_2 \cdot \textit{est-local-routing-area}(\mathcal{M}) \\ & + c_3 \cdot \textit{est-global-routing-area}(\mathcal{M}) \end{aligned} \tag{12.1}$$

where c_1, c_2, c_3 are constants,

$$\textit{est-local-routing-area}(\mathcal{M}) = \textit{min-track-width} \sum_{M \in \mathcal{M}} \textit{perimeter}(M) \cdot \textit{n-degree}(M) \tag{12.2}$$

and

$$\textit{est-global-routing-area}(\mathcal{M}) = \textit{min-track-width} \cdot n\sqrt{\textit{area}(\mathcal{M})} \tag{12.3}$$

The constants c_1, c_2, c_3 can be adjusted to reflect the type of chip being made (say, $c_1 = 1.3$, $c_2 = c_3 = .5$). Of course, these equations are intended only to yield rough estimates.

12.2 Pad Placement

After estimating the size and shape of the logic box, PI places the pads around the edge of the chip using a special-purpose routine. This section describes PI's pad placement heuristic in detail and discusses the major issues that were considered in the design of this heuristic.

input: **module-list**
output: **ordered-pad-list**

1. Initialize all module scores to 0. Initialize **ordered-pad-list** to the empty list. Let m be the length of **module-list**.
2. Repeat until **module-list** is empty:
   (a) Let M be any module on **module-list** of highest score. Delete M from **module-list**. If M is a pad, then push M onto **ordered-pad-list**.
   (b) Update module scores for all modules on the **module-list** that touch M. Specifically, for each terminal T_1 on M, for each terminal T_2 on $N = net(T_1)$, increment the module score for $module(T_2)$ by Δ, where $\Delta = m - t\text{-}length(N)$.

Figure 12.1: Pad-ordering heuristic

In placing the pads, PI is sensitive to special engineering concerns and to the interconnectivity of the pads with the rest of the modules. Pad placement affects logic module placement, since PI's algorithms for placing logic modules are sensitive to the context in which the logic modules are placed.

12.2.1 How PI Places Pad Modules

PI places the pads in two steps. First, PI orders the pads. Second, PI distributes the pads around one or more sides of the chip without altering the pad ordering.

The main idea of the pad-ordering heuristic is to group pads that are highly interconnected. The measure of connectivity is based on the entire circuit graph, and not just on the circuit subgraph whose nodes are pads. Also, the measure of connectivity weights small nets more strongly than large nets. Figure 12.1 gives the details of PI's pad ordering heuristic. The heuristic makes use of a *module-score* field for each module, and the heuristic actually computes an ordering on all of the modules. As with the algorithm for estimating chip size, the goal of this heuristic is more important than its details.

When PI distributes the pads around the edge of the estimated logic box, PI uses the fewest number of sides, thereby attempting to minimize chip area. If PI uses only two sides for pads, then PI places the pads on opposite sides of the chip. For each side used, PI creates a side box to contain all pads for that side. Within each side box, each pad is oriented according to its preferred orientation, as specified in PI's input. In the exceptional case that there are too many pads to fit around the logic

box, PI uniformly expands the logic box to make room for the pads.

Immediately after placing the logic modules, PI adjusts the placement of the side boxes. Neither the pad ordering nor the distribution of pads among the boxes is altered. Although not formally part of the pad placement routine, this final adjustment is needed to accommodate any change in the shape or size of the logic box that may have occurred during logic placement and to leave space for the power and ground wires.

12.2.2 Pad Placement Issues, as Seen by PI

During the design of the pad placement heuristic, the PI team focused on two major issues—the special concerns involving pads and the relationship between pad placement and logic module placement.

As with logic module placement, pad placement can affect chip area and wire length. But more so than with logic module placement, pad placement is greatly affected by numerous engineering concerns, including bonding requirements, power consumption, and thermal expansion. Pads often have greater power requirements than logic modules, and pads usually touch only a small number of nets. Since pads are the communication points of the chip, pad placement influences how the chip can be combined with other chips to form larger systems. These special considerations led the PI team to treat pad placement separately from logic module placement.

PI requires all pads to be placed along the chip's periphery. The main reason behind this decision was to conform to requirements imposed by some bonding companies. Also, putting pads on the edge of the chip makes power-ground routing to pads especially convenient. In addition, this decision leaves a "geometrically clean" problem of placing the logic modules in an unobstructed rectangular region.

An alternate strategy used by some chip designers is to put all pads in the center of the chip. The main purpose of this strategy is to reduce stress on the bonding wires due to thermal expansion of the chip. During the normal operation of a chip, the chip will expand and contract as it heats and cools. If all parts of the chip expand uniformly, movement is minimized at the chip's center.

The PI team decided to place the pads *before* placing the logic modules. We believed this decision is more flexible than placing the logic modules first, since it allows natural orderings of the pads to be preserved more easily. (When chips are designed to fit together with other chips, it is sometimes convenient to order the pads to simplify routing between chips. Such pad orderings might not be apparent from any single chip. To accommodate this situation, an option exists in PI for the user to specify some or all of the pad ordering.) Also, placing pads before placing logic modules leaves open the possibility of modifying pad placement after logic placement.

During its deliberations on pad placement, the PI team briefly discussed what type of algorithm might be effective for placing pads, if pad placement were performed after

logic placement. One promising idea was to use an iterative-improvement heuristic that would attempt to minimize the center of gravity of the nets, with terminals close to pads weighted more heavily than terminals far away from pads. Since PI places the pads before placing the logic modules, this heuristic was never developed in detail.

12.3 Top-Down Mincut Partitioning

The cornerstone of PI's placement algorithms is a top-down recursive mincut process that determines an approximate placement of the logic modules. This mincut process successively refines the initial logic tree into a binary tree whose leaves are the module boxes. The resulting placement hierarchy describes a recursive slicing of the logic box, each leaf rectangle of which corresponds to a distinct logic module. At each step, PI divides a rectangle in the slicing and correspondingly partitions the set of modules to be placed in that rectangle. By attempting to minimize the number of nets that have terminals in both divisions of the rectangle, PI reduces wirelength, facilitates routing, and simplifies the remaining mincut steps.

This section describes PI's mincut process in detail, focusing on how PI partitions a set of modules and on how PI partitions a rectangle.

12.3.1 Summary of Mincut Process

At the beginning of the mincut process, the logic tree consists of the root logic box together with its children, the module boxes. PI has already estimated the size of the logic box. During the mincut process, PI refines the logic tree in a breadth-first fashion. With each step, PI determines in greater and greater detail approximately how the logic modules should be placed.

During each step of the mincut process, PI operates on a sized node of the logic tree. The node represents an approximate placement of two or more modules intended to be placed within the node's rectangular extent. These modules are the node's children. PI partitions the node's children into two subsets, each with approximately the same total module area. PI also partitions the node's extent into two rectangles, one corresponding to each subset of modules. Each new rectangle is intended to contain the placement of its corresponding modules. By making the subsets roughly balanced in module area, PI reduces the complexity of the remaining mincut steps and uses chip area more efficiently.

In partitioning the modules, PI attempts to minimize the number of nets that touch both subsets. PI partitions the modules using the Fiduccia-Mattheyses implementation of the Kernighan-Lin graph partitioning heuristic, modified to take context into consideration. PI tries both vertical and horizontal partitions and chooses the partition it deems best as measured by a score function.

Because PI's graph partitioning heuristic is sensitive to the context in which the partition is made, horizontal and vertical partitions can yield different results. For the same reason, breadth-first and depth-first traversals of the logic tree can produce different placement hierarchies. By refining the logic tree in breadth-first order, PI makes rough decisions affecting large areas of the chip before making detailed local decisions.

At the end of each step, PI modifies the logic tree. PI creates two new nodes, one corresponding to each subset of modules. The modules in each subset become the children of their corresponding new node, and the new nodes become the children of the original node. At this point, PI declares the original node to be planned.

12.3.2 Partitioning the Modules

Input to each recursive step of the mincut process is a pair $(P, \mathcal{T})$, where P is a free and sized pi-box in the placement tree $\mathcal{T}$. The children of P are module boxes and there are at least two children. This section explains how PI partitions the children of P into two subsets.

Let $\mathcal{M}$ be the set of P's children. PI partitions $\mathcal{M}$ into two nonempty subsets $\mathcal{A} \cup \mathcal{B} = \mathcal{M}$ such that $\mathcal{A}$ and $\mathcal{B}$ have approximately equal module areas and such that the number of nets touching modules in both $\mathcal{A}$ and $\mathcal{B}$ is as small as possible. Any net with terminals in both subsets is said to be *cut by the partition*. For any partition $\mathcal{A} \cup \mathcal{B} = \mathcal{M}$, let *cost*(A, B) denote the number of nets that cut by the partition.

The Kernighan-Lin Heuristic

PI partitions the modules using a variation of a graph partitioning heuristic described by B. W. Kernighan and S. Lin [391,33]. This iterative-improvement algorithm begins with an initial partition which it then improves through a sequence of *passes*. PI carries out passes until the partition cost no longer decreases, up to a maximum number of passes (PI arbitrarily sets this maximum at 10).

Each pass consists of a sequence of *moves*, where a move reassigns a module from one side of the partition to the other. PI moves each module at most once during each pass. Following the Fiduccia-Mattheyses scheme, two successive moves can reassign modules from the same side. Thus, PI's notion of a move differs slightly from that described by Kernighan and Lin. To ensure that the partition is sufficiently balanced, each move must satisfy an *area-balance criterion*. Moves continue until all modules have been moved or until no module can be moved without violating the area balance criterion.

PI applies a greedy strategy to select the next module to move. Let the *gain of a module* be the decrease in partition cost that would result if the module were moved. Among all unmoved modules, PI selects the module of greatest gain, even if that

gain is negative. Moving modules with negative gains enables the heuristic search to proceed beyond locally optimal partitions. At the end of the pass, PI restores the best partition encountered during the pass.

The Fiduccia-Mattheyses Implementation of the Kernighan-Lin Heuristic

PI follows an implementation of the Kernighan-Lin heuristic due to Chuck Fiduccia and Robert Mattheyses [388]. This implementation uses a simple data structure to achieve a worst-case running time per pass that is linear in the number of terminals in $\mathcal{M}$.

The following questions arise in the implementation of the Kernighan-Lin heuristic. What is the initial partition? How is the next module selected? How are the module gains updated after each move? What area-balance criterion is used? How is the best partition of the pass restored?

PI creates an initial partition as follows. As suggested by Fiduccia and Mattheyses, PI begins with all modules on one side of the partition and successively moves modules from that side until the area-balance criterion is satisfied. At each step, PI moves a module of maximum gain. Although better results could probably be obtained by repeating the Kernighan-Lin heuristic on several different randomly selected initial partitions [360], for simplicity, the initial implementation of PI follows the simple deterministic scheme described above.

To select the next module efficiently, PI stores the modules in an array indexed by gain. Each element of the array is a doubly linked list of module boxes with the same gain. To locate each module within the data structure quickly, each module box points to the module's location within the appropriate linked list. PI uses a separate array for each side of the partition, thus providing flexibility to support a range of strategies for selecting the next module to be moved.

PI considers a partition balanced if and only if the sums of the module areas for each side differ by at most twice the area of the largest module. This area-balance criterion ensures that all modules can be moved within each pass.

To reconstruct the best partition found during the pass, PI simply records the sequence of moves made during the pass.

External Connections

In computing partition costs, PI considers not only how nets touch modules within $\mathcal{M}$ but also how nets touching $\mathcal{M}$ connect with modules not in $\mathcal{M}$. This extension of the Kernighan-Lin heuristic makes PI sensitive to the context in which partitions are made. A similar heuristic was independently developed by Dunlop and Kernighan [387].

An *external connection of the pi-box P* is a pin that is not on any descendant

module of P, but that belongs to a net touching P.[1] At the beginning of PI's mincut refinement of P, PI determines for each net N touching P whether P has any external connections on N that lie outside of P's extent. To locate these external connections in the approximate placement of the logic modules, PI recomputes the vpins of the logic box excluding all pins on children of P. External connections known to lie outside of PI's extent contribute to partition costs.

Without loss of generality, assume that PI is working on a horizontal partition; that is, PI will place one subset of modules above the other across a line parallel to the x-axis of the chip. Further assume that $\mathcal{A}$ is the subset of modules to be placed above this line and $\mathcal{B}$ is the subset to be placed below the line. For each net N touching P, if there is at least one external connection on N north of P's extent, then PI computes partition costs as if there were an additional child of P touching N permanently located in $\mathcal{A}$. Similarly, if there is at least one external connection south of P's extent, then PI proceeds as if there were an additional child of P touching N permanently located in $\mathcal{B}$.

If there are external connections on N both north and south of P, then the effects of these external connections cancel. However, if there are external connections only north of P, then while PI partitions P's children, PI gives preference to place modules that touch N in the upper subset $\mathcal{A}$ (see figure 12.2).

12.3.3 Drawing the Floorplan

As part of each mincut step, PI partitions the extent of the current node into two rectangles, one corresponding to each subset of modules computed during module partitioning. Each rectangle is intended to contain the placement of its corresponding subset of modules. The floorplan, which is determined by either a vertical or horizontal division of the node's extent, influences remaining mincut steps as well as the module orientation and hardening phases.

Let P be the current node being refined. During the mincut refinement of P, PI partitions P's children and divides P's extent twice—once using a horizontal partition and once using a vertical partition. Among the two competing partitions, PI chooses the one it deems best as measured by a score function.

Slicing the Current Rectangle

PI divides P's extent so that the ratio of the resulting rectangle areas is equal to the ratio of the areas of the corresponding subsets of modules. The intent of this simple method is to allocate the available layout area in proportion to the relative needs of the two subsets of modules. This method, however, does not take routing area into consideration.

[1] Recall that *a net touches a pi-box* if the net has a terminal on a descendant module of the pi-box.

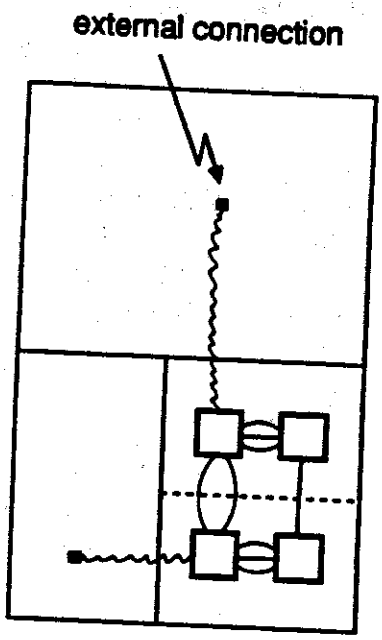


Figure 12.2: Partitioning modules in the context of an approximate placement

Determining Partition Orientation

PI evaluates each partition by computing a simple score function that takes three factors into consideration—cost of module partition, area balance of module partition, and shape of rectangles in floorplan. A fourth term measuring how well the modules fit inside of their corresponding rectangles was considered but never implemented.

Let $\mathcal{A}$ and $\mathcal{B}$ be the subsets of modules computed by the mincut heuristic, and let R_A and R_B denote the rectangles in the floorplan that correspond to $\mathcal{A}$ and $\mathcal{B}$ respectively. PI computes partition score as a linear combination of three normalized terms.

$$\begin{aligned} \textit{partition-score}(\mathcal{A}, \mathcal{B}, R_A, R_B, \mathcal{T}) \;=\; & c_1 \cdot \textit{nets-cut-term}(\mathcal{A}, \mathcal{B}, \mathcal{T}) \\ & + c_2 \cdot \textit{area-balance-term}(\mathcal{A}, \mathcal{B}) \\ & + c_3 \cdot \textit{aspect-ratio-term}(R_A, R_B) \end{aligned} \tag{12.4}$$

where c_1, c_2, and c_3 are constants and $\mathcal{T}$ is the placement tree (PI uses $c_1 = 1$, $c_2 = .3$, and $c_3 = .3$).

The nets cut term measures how well the mincut heuristic reduced partition cost. PI computes this term as $1-(k-\hat{k})/(n-\hat{k})$, where n is the number of nets touching $\mathcal{A}$ or $\mathcal{B}$, k is the partition cost (considering external connections), and $\hat{k}$ is the number of nets necessarily cut by external connections. We say that a net is necessarily cut by an external connection if and only if the net has an external connection on both sides of the current partition.

The area balance term measures how well the subsets $\mathcal{A}$ and $\mathcal{B}$ are balanced in total module area. PI computes this term as $\alpha_{min}/\alpha_{max}$, where $\alpha_{min} = \min\{\textit{area}(\mathcal{A}), \textit{area}(\mathcal{B})\}$ and $\alpha_{max} = \max\{\textit{area}(\mathcal{A}), \textit{area}(\mathcal{B})\}$.

The aspect ratio term measures the shapes of the rectangles R_A and R_B, penalizing long, skinny shapes. PI computes this term as $.5(\textit{aspect-ratio}(R_A) + \textit{aspect-ratio}(R_B))$.

12.4 Module Orientation

After computing a recursive mincut decomposition of the placement tree, PI determines the orientation of each module; that is, PI determines how each logic module should be flipped and rotated around its center. Module orientations affect how much wire is required for routing and how well modules fit together. PI's orientation step acts as a local, fine-tuning of the coarse approximate placement produced by mincut. This section explains how PI orients the logic modules.

The goal of PI's orientation step is to determine an orientation of the modules that minimizes layout cost. PI's heuristic approach is to orient each module, one after the other, separately minimizing estimated layout cost for each module.

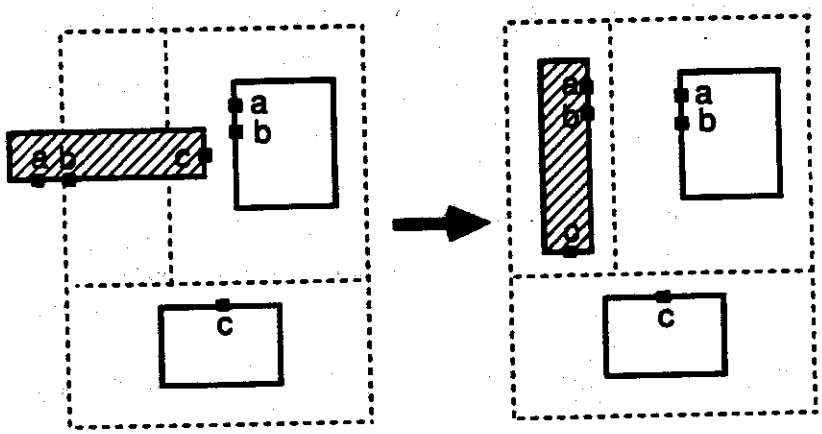


Figure 12.3: Module orientation in the context of a placement tree

12.4.1 Orientation After Mincut

After PI computes a complete mincut decomposition of the logic tree, each module box is the child of a unary box. At this point, only the module boxes are hardened.

PI orients each module, one after the other, in a postorder traversal of the placement tree. To preserve the orientations of internal pi-boxes produced during the mincut phase, PI orients only module boxes (and not supermodules) during its orientation step.

To orient a module box, PI carries out the following steps. First, PI translates the module box so that its center coincides with its parent's center. Then, PI tries all eight possible orthogonal transforms. For each transform, PI applies the transform to the module box and computes a cost function that estimates wire length and wasted area. Finally, PI selects the transform that minimizes the cost function (see figure 12.3).

Currently, PI orients each module only once. Since the orientation of any one module can affect the orientation of any other module, better results might be obtained by repeating the orientation process several times. Orienting the modules in a randomly chosen order might also yield better results.

12.4.2 PI's Orientation Cost Function

Input to PI's orientation refinement is a pair $(P, \mathcal{T})$, where P is a sized unary pi-box whose only child is hardened, and $\mathcal{T}$ is the placement tree. Let A be child of P. To orient A, PI minimizes the following orientation cost function, which is a linear combination of estimated wire-area and wasted area terms. The wire area term includes a contribution for each net that touches A. Specifically,

$$\textit{orientation-cost}(A, P, \mathcal{T}) = c_1 \cdot \textit{est-wire-area}(A, \mathcal{T}) + c_2 \cdot \textit{wasted-area}(A, P) \quad (12.5)$$

where

$$\textit{est-wire-area}(A, \mathcal{T}) = \textit{min-track-width} \sum_{t_0 \in \text{terminal-list}(A)} \textit{est-wire-length}(t_0, \textit{net}(t_0)) \quad (12.6)$$

and c_1 and c_2 are constants (PI uses $c_1 = c_2 = .5$), and $\textit{wasted-area}(A, P)$ is the area of A's extent that protrudes outside of P's extent. The quantity $\textit{est-wire-length}(t_0, \textit{net}(t_0))$ measures the wire length associated with the net that touches terminal t_0.

Let $N = \textit{net}(t_0)$. PI computes $\textit{est-wire-length}(t_0, N)$ as a weighted sum of the distances between t_0 and the other terminals on N. Terminals close to t_0 are weighted more heavily than terminals far away from t_0. The intent of this weighting is to align the pins of A on N with nearby pins on N. PI computes

$$\textit{est-wire-length}(t_0, N) = \sum_{t \in \text{terminal-list}(N)} \textit{weight}_{t_0}(t) \cdot \textit{dist}(t_0, t) \quad (12.7)$$

where $\textit{weight}_{t_0}(t)$ is the weight of terminal t.

Let $r = \textit{t-length}(N)$ and let $(\hat{t}_0, \hat{t}_1, \ldots, \hat{t}_{r-1} = t_0)$ be the terminals of N, listed in decreasing distance from t_0. PI uses a normalized geometric weighting scheme in which $\textit{weight}_{t_0}(t_0) = 0$ and $\textit{weight}_{t_0}(\hat{t}_i) = 2^i/(2^{r-1} - 1)$, for all $0 \leq i < r - 1$.

To compute the orientation score efficiently, PI must cope with a few more details. The first problem is that PI must express the pin positions in a common coordinate system. At the beginning of the orientation process, PI computes the center of each pin in the chip box's coordinate system. For each candidate orientation, only the locations of the pins on A are recomputed. The second problem is to compute $\textit{est-wire-length}(t_0, N)$. For each terminal t_0 on A, PI sorts the terminals on N by decreasing distance from t_0 and then computes the weight for each terminal.[2]

[2]For long nets, PI can optionally compute the wire-length term considering only the k closest terminals on N to t_0, for some moderate k (say, $k = 10$).

12.5 Bottom-Up Hardening

After finishing its mincut and orientation steps, PI transforms the approximate, planned placement into an exact legal placement. Known as hardening, this transformation aligns pins and leaves space for routing.

PI hardens the approximate placement in a postorder traversal of the logic tree. To harden any planned pi-box, PI determines how the children of the pi-box are to be placed next to each other. PI solves this problem of placing two pi-boxes next to each other as an optimization problem with routing constraints. During the hardening process, PI preserves the orientations of modules and pi-boxes produced by the orientation and mincut refinements.

We now explain in detail how PI hardens any planned pi-box. Section 12.5.1 summarizes PI's hardening process. Section 12.5.2 explains how PI computes the displacements between the two children of a planned binary box. Section 12.5.3 gives the details of PI's hardening calculation for the special case in which the children are modules, and section 12.5.4 describes the general case in which the children are hardened pi-boxes.

12.5.1 Summary of Hardening Process

Input to the hardener is a planned binary pi-box P whose children, A and B, are hardened pi-boxes. The hardener determines exactly how A and B should be placed relative to each other.

PI specifies the relative placement of P's children in terms of an offset and separation. Without loss of generality, assume that P's orientation is horizontal and that A is to be placed above B in P. Thus, the hardening problem is to place A and B across a horizontal line. The *offset* is the horizontal distance between the left edges of (the extents of) A and B, where positive offsets move B to the right of A. The *separation* is the vertical distance between A and B (see figure 12.4).

After computing an offset and separation, PI translates A and B appropriately and modifies P's extent to fit tightly around A and B. At this point, P is declared hardened.

In addition to producing a legal placement, the hardener also leaves space for routing. Subject to routing constraints, the hardener attempts to minimize both the area of the smallest box bounding A and B and the estimated wire area required to route nets between A and B. But as illustrated in figure 12.5, hardening involves inherent tradeoffs between minimizing area and minimizing wire length. PI resolves these tradeoffs by minimizing bounding area, where bounding area includes space left for routing.

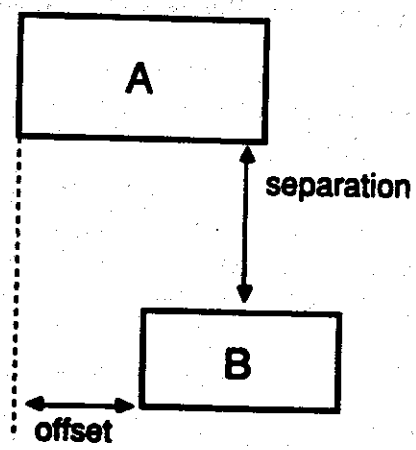


Figure 12.4: Hardening computes an offset and separation between two pi-boxes

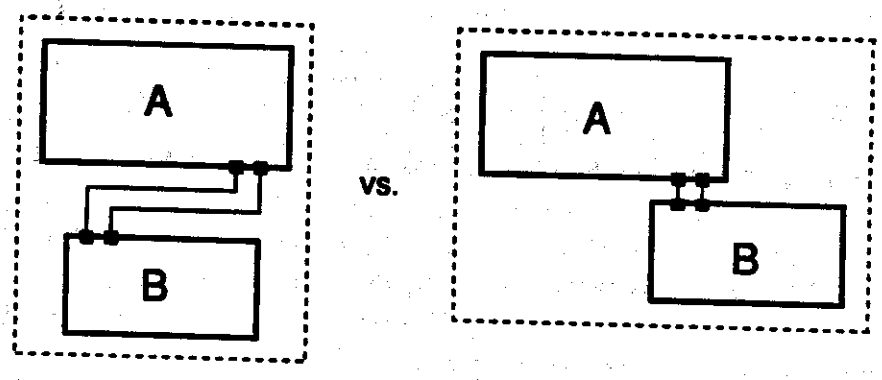


Figure 12.5: Hardening involves tradeoffs between area and wire length

12.5.2 How PI Computes Separation and Offset

PI computes the separation and offset between two pi-boxes by expressing separation as a function of offset. Then, PI finds the offset that minimizes the bounding area of the two pi-boxes. The separation, which is based on wire length estimates, attempts to leave enough room for routing between the pi-boxes. Thus, PI casts the hardening problem as an optimization problem with one degree of freedom—offset.

PI computes separation by estimating the required number of horizontal routing tracks needed between the two pi-boxes. For any offset x, PI computes

$$\textit{separation}(x) = \textit{min-track-width} \cdot \textit{required-tracks}(x) \tag{12.8}$$

where *required-tracks*(x) is PI's estimate of the required number of horizontal routing tracks. This quantity depends on PI's estimate of the required amount of horizontal wiring. More specifically, for any offset x,

$$\textit{required-tracks}(x) = \frac{c \cdot \textit{hor-wire-length}(x)}{\max\{x_A, x_B\}} + \textit{extra-tracks} \tag{12.9}$$

where x_A and x_B are the horizontal dimensions of A and B respectively. In equation 12.9, *hor-wire-length*(x) estimates the horizontal wire length required for routing between A and B all nets that touch both pi-boxes. The terms c and *extra-tracks* are constants (PI uses $c = 1.6$ and *extra-tracks* $= 5$). The *extra-tracks* term adds space for additional nets that may be routed between the pi-boxes.[3]

For any offset x, provided *hor-wire-length*(x) is known, bounding area can now be computed as

$$\textit{bounding-area}(x) = \textit{channel-width}(x) \cdot [y_A + \textit{separation}(x) + y_B] \tag{12.10}$$

where y_A and y_B denote the heights of A and B respectively, and *channel-width*(x) is the horizontal distance between the outer edges of A and B.

Let $\hat{x}$ be the offset that minimizes bounding area. From equations 12.8–12.10, it follows that, if *hor-wire-length*$(\cdot)$ is convex, then so is *bounding-area*$(\cdot)$. Fortunately, *hor-wire-length*$(\cdot)$ is a convex function. Hence $\hat{x}$ can be found by binary search.

For one special case, there is a short-cut method for computing $\hat{x}$. The special case is when $\hat{x}$ lies within the range of offsets for which *channel-width*$(\cdot)$ does not depend on offset. This situation happens, for example, when B is much shorter than A. For this special case, the same offset minimizes *bounding-area*$(\cdot)$ and *hor-wire-length*$(\cdot)$. Since the offset that minimizes *hor-wire-length*$(\cdot)$ can be found by a short-cut method, so can $\hat{x}$.

The next two sections describe how PI computes *hor-wire-length*$(\cdot)$ and $\hat{x}$.

[3] An alternate approach might compute *extra-tracks* as a function of the input circuit graph or as a function of the height of P in the placement tree.

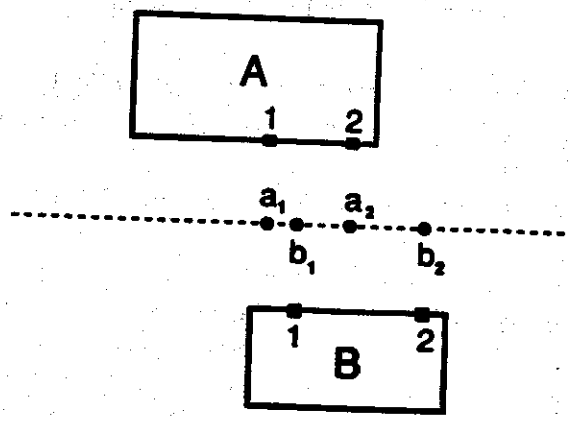


Figure 12.6: Special case of hardening: Placing two modules across a channel

12.5.3 Placing Two Modules Across a Channel

This section completes the description of PI's hardening process for the special case in which the pi-box, P, to be hardened is the father of two module boxes A and B. Throughout this chapter, we will continue to assume that PI is placing A above B across a horizontal line. As explained in the previous section, PI computes the separation between A and B as a function of the estimated amount of horizontal wiring needed to route between the modules. This estimate depends on the distance B is offset from A. It remains to be explained how PI estimates horizontal wire length and how PI finds the offset, $\hat{x}$, that minimizes bounding area. This section explains how PI calculates these values in the special case in which all pins lie along the channel separating the modules and each net touching A and B has exactly two pins (see figure 12.6). The next section explains the general case.

Let $N_1, \ldots, N_n$ be the n nets that touch both A and B. Also, let $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ be the horizontal coordinates of the centers of the pins on A and B respectively, expressed in P's coordinate system. For each $1 \leq i \leq n$, net N_i contains the pins with coordinates a_i and b_i.

For any offset x, PI estimates required horizontal wire length as the sum of the required horizontal wire length for each net. For each $1 \leq i \leq n$, the required horizontal wire length for net N_i is $|x_i^* - x|$, where $x_i^* = a_i - b_i$ is the offset that aligns the pins of net N_i. Thus, for any offset x, for this special case of hardening, the required horizontal wire length is

$$\textit{hor-wire-length}(x) = \sum_{i=1}^{n} |x_i^* - x| \tag{12.11}$$

Let x^* be the offset that minimizes *hor-wire-length*$(\cdot)$, and let $x^*_{median} = \text{median}\{x_1^*, \ldots, x_n^*\}$. Equation 12.11 defines a convex function that takes on its minimum value at x^*_{median}. To see this fact, consider any offset $x \neq x^*_{median}$. By moving x toward x^*_{median}, total horizontal wire length decreases. Total horizontal wire length decreases because horizontal wire length for each net increases or decreases by the same amount, and because horizontal wire length for a majority of the nets decreases. Because x^*_{median} can be computed in $O(n)$ steps, so can x^*, even though computing *hor-wire-length*$(\cdot)$ also requires $O(n)$ steps. This observation was made by C. S. Chow, and, independently, by Dolev, Karplus, and Siegel [421].

Let x_L and x_R be the offsets that align respectively the left edges and right edges of A and B. In addition, let $x^*_{min} = \min\{x_1^*, \ldots, x_n^*\}$ and $x^*_{max} = \max\{x_1^*, \ldots, x_n^*\}$.

If $x_L \leq x^*_{min}$ and $x^*_{max} \leq x_R$, then *channel-width*(x) = *channel-width*(x_L) for all offsets x such that $x_L \leq x \leq x_R$. Hence, in this case, $\hat{x} = x^*$. But in general, $\hat{x}$ might be none of $x_1^*, x_2^*, \ldots, x_n^*, x_L, x_R$. However, since *hor-wire-length*$(\cdot)$ is a convex function, so is *bounding-area*$(\cdot)$. Therefore, $\hat{x}$ can always be found by binary search in the interval $[x_{min}, x_{max}]$, where $x_{min} = \min\{x_L, x^*_{min}\}$ and $x_{max} = \min\{x_R, x^*_{max}\}$.

PI calculates offset as follows. If $x_L \leq x^*_{min}$ and $x^*_{max} \leq x_R$, then PI computes $\hat{x} = x^*$ using the $O(n)$ time short-cut method which finds x^*_{median}. Otherwise, PI computes $\hat{x}$ by golden-section search [28] in the interval $[x_{min}, x_{max}]$. PI uses golden-section search rather than binary search to reduce the number of times bounding area has to calculated, since each bounding area calculation requires $O(n)$ steps. Actually, to enhance pin alignments and to increase speed, PI restricts its search to the "critical offsets" $x_1^*, x_2^*, \ldots, x_n^*, x_L, x_R$, even though $\hat{x}$ might not be any of these points.

12.5.4 Placing Two Hardened Pi-Boxes Across a Channel

This section completes the explanation of the general case of hardening in which the pi-box to be planned, P, is the parent of two arbitrary hardened pi-boxes A and B. For this case, nets may have multiple terminals, and pins may appear anywhere within the extents of the two pi-boxes. PI uses a natural generalization of its technique for placing two module boxes across a channel. In this general case, PI works with extents of nets (*i.e.* vpins) rather than with pins.

Assume A and B are arbitrary hardened pi-boxes. Instead of dealing with the horizontal coordinates of the pin centers of pins on two-pin nets, PI now deals with the projections of vpins onto the horizontal line separating A and B. For each net N_i touching A and B, PI computes an interval $[a_i, a_i']$ that is the horizontal extent of the vpin for net N_i in A. Similarly, PI computes intervals $[b_i, b_i']$ for the vpins in B.

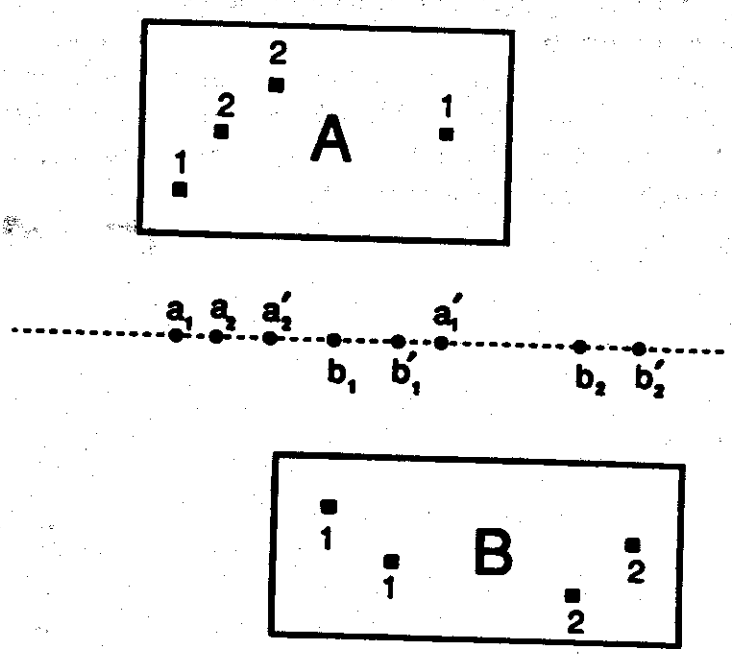


Figure 12.7: General case of hardening: Placing two pi-boxes across a channel

PI expresses each of these intervals in P's coordinate system (see figure 12.7). Thus, instead of treating pins as points, PI now treats vpins as intervals.

For each net N_i touching A and B, PI computes a range of offsets $R_i = [u_i, v_i]$ for which the vpin intervals $[a_i, a_i']$ and $[b_i, b_i']$ overlap. Instead of there being a single optimal offset for each net, there is now a range of optimal offsets for each net. Using these optimal offset ranges, for any offset x, PI estimates required horizontal wire length as

$$\textit{hor-wire-length}(x) = \sum_{i=1}^{n} \textit{dist}(x, R_i) \tag{12.12}$$

where $\textit{dist}(x, R_i)$ is the distance between x and the nearest point in R_i.

Let $u_{min} = \min\{u_1, u_2, \ldots, u_n\}$ and let $v_{max} = \max\{u_1, u_2, \ldots, u_n\}$. Also, let $x_{min} = \min\{u_{min}, x_L\}$ and $x_{max} = \max\{v_{max}, x_R\}$.

As in the special case of hardening, *hor-wire-length*$(\cdot)$ is still a convex function of offset. Hence, bounding area is a convex function of offset, and offset, $\hat{x}$, that minimizes bounding area can be found by binary search. Moreover, if $x_L \leq u_{min}$ and $v_{max} \leq x_R$, then $\hat{x}$ is the offset that minimizes *hor-wire-length*$(\cdot)$, which is the median of the endpoints $u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n$.

If $x_L \leq u^*_{min}$ and $v_{max} \leq x_R$, then PI computes $\hat{x} = x^*$ using an $O(n)$ time shortcut method which computes the median of the endpoints $u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n$. Otherwise, PI computes $\hat{x}$ by golden-section search in the interval $[x_{min}, x_{max}]$. As before, PI uses Fibonacci search to reduce the number of times bounding area has to be calculated. Although $\hat{x}$ is not necessarily one of $x_L, x_R, u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n$, PI restricts its search to these critical points.

Chapter 13

Extensions to the PI System

During its deliberations, the PI team considered several layout algorithms that were never incorporated into the PI System. The first three sections of this chapter briefly discuss some of these extensions and alternate approaches. Section 13.1 describes two additional placement strategies that can be incorporated into PI's placement framework. Section 13.2 describes a new crossing placement algorithm. Section 13.3 summarizes four ideas for a new switch-box channel router. The chapter concludes with a description of an extended version of PI developed at the General Electric Research and Development Center.

13.1 Additional Placement Algorithms

One early design of PI's placement algorithms envisioned that the process of refining the placement tree would include two additional moves known as *bottom-up pairing* and *combinatorial search*. *Bottom-up pairing* identifies two hardened pi-boxes that "fit together well" and places them next to each other. *Combinatorial search* finds an exact placement for a small number of modules by exhaustively searching a subspace of all possible placements. Each of these additional refinements can be conveniently implemented within PI's placement framework.

Although bottom-up pairing does not enlarge the set of placements possible under PI's current refinements, this additional heuristic would help PI find better placements. Unlike bottom-up pairing, combinatorial search expands the set of placements possible under PI's current refinements because combinatorial search is not restricted to placements based on slicings.

For a variety of reasons, bottom-up pairing and combinatorial search were never implemented. Most importantly, the current refinements were implemented first and seemed to work well. Since mincut tends to pair modules that are highly interconnected, and since bottom-up pairing does not expand the set of possible placements, the PI team doubted that bottom-up pairing would make a significant improvement.

Also, the idea of combinatorial search appears promising, but many important questions remain to be worked out.

This section describes some preliminary ideas for implementing the bottom-up pairing and combinatorial search refinements. The section begins with a suggestion for how these refinements can be incorporated into PI's placement process.

13.1.1 An Alternate Control Strategy

This section describes one strategy through which the bottom-up pairing and combinatorial search refinements can be incorporated into the placement process. Under this strategy, PI refines the placement tree in the following five steps:

First, PI repeatedly applies the bottom-up pairing refinement to pair modules that fit together very well. This process continues as long as well-matched pairs are found, as identified by a pairing score computed by the bottom-up pairing refinement.

Second, PI performs top-down mincut partitioning until each pi-box of the placement tree has only a few children. Specifically, PI applies mincut only to pi-boxes with at least Υ children, where $\Upsilon \geq 4$ is a parameter that depends on the resources of the computer running PI. The combinatorial search refinement will complete the refinement of the pi-boxes not processed by mincut. Since some modules might already be paired, the mincut process must treat hardened pairs as if they were modules.

Third, PI orients the modules and the pairs produced by the bottom-up pairing refinement. Specifically, PI orients each hardened pi-box that is an only child. For this step, the orientation refinement treats each hardened pair as if it were a module.

Fourth, PI applies combinatorial search to complete the work left by the mincut refinement. Specifically, PI determines the exact placements of all leaves of the planned subtree of the logic tree.

Fifth, PI applies the bottom-up hardening refinement to harden all remaining unhardened nodes of the placement tree.

13.1.2 Bottom-Up Pairing

Some chips have pairs of modules that can be placed next to each in a very natural way. This situation often arises, for example, with a PLA and its driver. The bottom-up pairing refinement identifies such module pairs (or, more generally, hardened pi-boxes) and determines exactly how the modules should be placed next to each other. This section discusses some preliminary thoughts on how PI might perform bottom-up pairing. Some of the ideas in this section were originally developed by C. S. Chow.

Given a pi-box P that has at least three hardened children, bottom-up pairing selects two hardened children of P and places them next to each other. Let A and B denote these paired children. PI creates a new pi-box C and makes C a child of P.

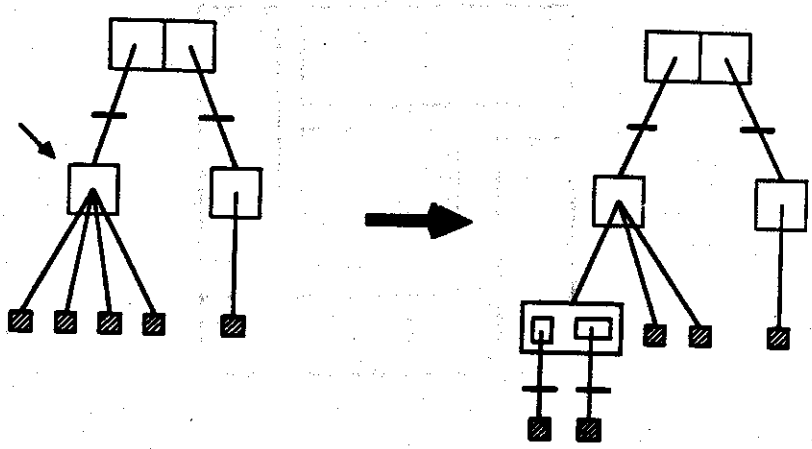


Figure 13.1: Bottom-up pairing refinement

Pi-boxes A and B become children of C and are removed as children of P. During bottom-up pairing, PI translates and orients A and B, leaving room for routing. PI sizes C to fit tightly around A and B, and C is declared hardened (see figure 13.1).

One approach is to compute for each pair of modules a normalized *pairing score* that measures how well the modules fit together. The two modules with the highest pairing score are placed next to each other.

Since bottom-up pairing deals with exact placements, it is important that the pairing score be sensitive to the orientations and relative positions of the two modules under consideration. Therefore, the pairing score is computed for each of the 32 different relative positions and orientations of the modules.[1] The pairing score for each pair of modules is taken to be the maximum of the pairing scores computed for all possible relative positions and orientations. To place the selected pair of modules next to each other, PI applies its hardening refinement, using the relative position and orientation that produced the highest pairing score.

To speed up the pairing selection process, PI could use pruning techniques that quickly reject certain pairs or relative positions of modules. Pruning of pairs might be based on interconnectivity; pruning of relative positions might be based on geometric factors.

For any pair of modules, for any relative position and orientation, PI computes the pairing score as follows. First, PI hardens the pair of modules using the hardening

[1] Given any two modules A and B, there are exactly 8 ways to flip and rotate A, and there are exactly 4 ways to rotate B. Together, these operations describe the 32 possible relative positions and orientations of A and B.

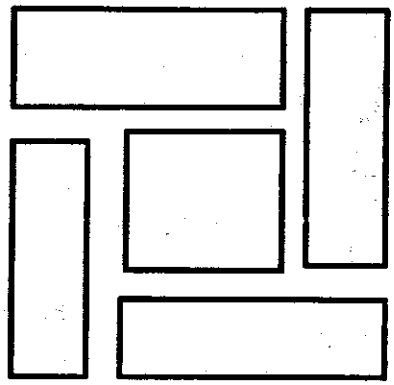


Figure 13.2: A nonslicing placement

refinement. Second, PI evaluates the hardened pair. Such a score might be computed as a weighted sum of terms based on the following considerations: layout density of bounding box (calculated from module area and estimated wire area), number of nets touching the two pi-boxes (with short nets weighted more heavily than long nets), and estimated wire length for routing between the two paired pi-boxes (normalized with respect to number of nets).

13.1.3 Combinatorial Search

The combinatorial search refinement computes an exact placement of a small number of modules. This refinement casts the placement problem as a search problem over a finite subset of possible placements. PI finds an optimum placement by evaluating each placement in the search space. Although combinatorial search could also be used in a bottom-up pairing strategy, the PI team intended the combinatorial search refinement to be used on small partial placements produced by mincut. This section summarizes some preliminary thoughts on how combinatorial search might be carried out.

One important reason for using a combinatorial search refinement is to enlarge the set of placements that PI can produce. In contrast with PI's current mincut and bottom-up pairing refinements, combinatorial search is not limited to slicings. Although all placements of three or fewer modules are slicings, this is not true when there are four or more modules (see figure 13.2). Another reason for using combinatorial search is to exploit the power of exhaustive search. To place a small number of modules, it may be feasible to try all "essentially different" placements.

Casting the placement problem as a discrete search problem is of additional in-

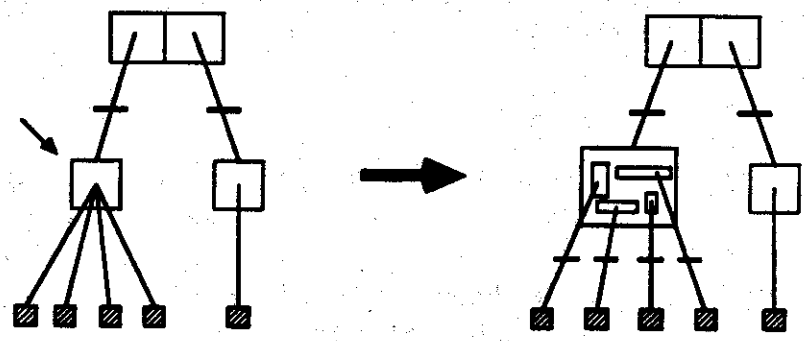


Figure 13.3: Combinatorial search refinement

terest because such a formulation could lead to general-purpose placement strategies based on heuristic search techniques.

How Combinatorial Search Works

The combinatorial search refinement determines an exact placement of a set of modules (or, more generally, hardened pi-boxes) in the context of a placement tree. Input to the combinatorial search refinement is a a pair $(P, \mathcal{T})$, where P is a sized pi-box P in the placement tree $\mathcal{T}$. P must have at least three children, and each child of P must be hardened.[2] During combinatorial search, PI determines the exact relative placements and orientations of P's children, leaving space for routing. PI translates and orients the children appropriately and modifies P's extent to fit tightly around the children. Finally, P is declared hardened and the structure of $\mathcal{T}$ is unaltered (see figure 13.3).

Combinatorial search works by exhaustively searching a finite subset of placements. Since there are infinitely many legal placements, PI limits its search to a restricted subset of placements. Moreover, the subset is chosen so that it includes one representative for each "essentially different" placement.

For each placement considered in the search, PI hardens the placement and computes a *combinatorial search score*. The search returns the best hardened placement, as measured by the score. Since PI's hardener works only on slicings, a more general hardener would have to be created.

[2] If P has only two children, then the bottom-up pairing refinement can be used to place the children.

The combinatorial search score of the hardened placement can be based on the following considerations: estimated wire length (including both wiring within the placement and wiring from the placement to the rest of the chip), bounding area, and how well the placement fits into P's extent.

Defining the Search Space

The basis of the combinatorial search refinement is to search over a restricted, yet representative, subset of placements. This section suggests several tentative ideas for defining the search space.

One approach is to divide the set of legal placements into equivalency classes. The search space consists of one representative from each equivalency class. Within each equivalency class, hardening produces the same result. Of course, it should be possible to enumerate elements in the search space efficiently.

More specifically, the search space might be based on one of the following ideas: topologically distinct partitions of a rectangle, Voronoi diagrams [356,354], adjacency graphs (as defined by nearest neighbors), adjacency graphs (as defined by line of sight), or lexicographic orderings of corner points.

13.2 A New Crossing Placement Algorithm

A new crossing-placement algorithm was designed, but never implemented. This new algorithm performs crossing placement in three stages: *crossing ordering*, *crossing positioning*, and *crossing layer assignment*. The crossing ordering stage determines for each channel edge the order in which nets cross that edge. The crossing positioning stage calculates the exact position of each crossing. The crossing ordering stage assigns a layer to each crossing.

The new crossing ordering algorithm uses a greedy strategy developed by Rivest and Fiduccia that proceeds channel by channel [451]. For the special case of two-point nets, this algorithm minimizes the number of forced crossovers.

The new crossing position algorithm minimizes estimated wire length through formulating the crossing position problem as a linear programming problem. This algorithm is due to Mark Novick, who also proved some additional properties about the new Rivest-Fiduccia crossing ordering algorithm [448].

13.3 Ideas for Channel Routing

The main ideas behind PI's three switch-box channel routers are storing a library of routing patterns, scanning the channel from one side to the other, and routing the strands one at a time using a minimum-cost Steiner tree heuristic. Toward the

end of the project, the PI team considered designing an additional router. Although several promising ideas were discussed, a fourth router was never implemented. This section briefly describes the four major ideas that were considered by the PI team for building a new switch-box channel router. These ideas are heuristic search, routing to a grid, divide-and-conquer, and dilation-compaction.

Heuristic search routing treats channel routing as a search problem. Channel routing takes place through a sequence of moves, where each move routes one strand. As with the Lee router, heuristic search routing processes strands one at a time. But unlike the Lee router, backtracking, look-a-head, and other standard heuristics guide the search.

The *routing-to-a-grid* strategy first constructs a grid in the interior of the channel. The grid can have either regular or irregular spacings. Then, each crossing is routed to the grid in such a way that the remaining routing within the grid is simple.

Divide-and-conquer routing, sometimes called *hierarchical routing* [412,413], recursively divides the channel into subchannels until the subchannels can be easily routed. One idea considered for PI is to use PI's crossing-placement algorithm as the mechanism for defining the subproblems. Specifically, at each step, the channel is divided into two rectangles by a carefully chosen dividing line. If any strands are forced to cross the dividing line, then crossing placement is performed along the dividing line to determine the locations of these crossings.

Dilation-compaction routing works by expanding the channel, routing the expanded channel, and finally compacting the routed channel. One motivation behind this strategy is to separate the topological aspects of routing from the geometric concerns. The final compaction step attempts to map the solution of the expanded problem into a planar-equivalent routing that fits into the geometrically restricted channel. This method appears especially promising given the existence of good planar-compaction algorithms, such as the one recently proposed by Miller Maley [442]. According to Maley, researchers at Bell Laboratories have experimented with this style of channel routing [442].

13.4 PI at General Electric

The General Electric Research and Development Center (GE) in Schenectady, New York, modified PI into an extended system which they call 2PI. This section briefly describes what extensions GE added to PI and summarizes GE's experiences with 2PI.[3]

GE became interested in the PI System for routing custom VLSI chips. Since

[3]The information in this section is based on an August 7, 1985 telephone conversation with Dr. Robert M. Mattheyses and on private correspondence, dated October 8, 1985, from Ross Stenstrom and Robert Mattheyses.

most of GE's applications involved highly structured placements, GE preferred to place its chips by hand. Consequently, GE planned to use PI exclusively as a router. Initially, GE intended to use MIT's prototype system, with minor modifications, as production software.

13.4.1 How 2PI Extends PI

GE developed a new switch-box router and further modified PI in several ways. This work was carried out by Robert M. Mattheyses and Ross Stenstrom, with brief programming help from a few other people. GE's 2PI system extends PI in the following ways:

- The user can define *busses* (*i.e.* bundles of nets) each of which are treated as a single net during global routing.
- To facilitate the routing of critical nets, the user can specify a *priority* for each net. The priority of each net affects the order or manner in which the net is routed.[4]
- In 2PI, the user can intervene to a greater extent during various steps of the layout process than with PI. For example, the user can adjust placements by moving modules. (Specifically, the user can move any module by "pushing" it with a mouse.[5]) Also, the user can select any channel with the mouse and order 2PI to route the selected channel individually.
- 2PI uses a new switch-box channel router. The new GE router is similar in flavor to PI's slice router, but follows the original greedy router design [450] more closely. According to Stenstrom and Mattheyses, the new router works much more effectively than PI's slice router. To adapt the greedy router strategy to the switch-box context, the new router scans the channel in two passes, the first pass proceeding left to right, the second pass proceeding right to left [429].
- 2PI uses a regular global grid and forces all layout features, including the input pins, to be placed along this grid.

13.4.2 Experimental Results

Using their version of PI, GE successfully routed the signal wires on two test chips. Power and ground wiring was supplied only on the second chip.

[4]The PI System has a provision for assigning priorities to nets, but this provision is currently used only by the global router.

[5]A *mouse* is an input device through which a user can point to locations on a display screen.

The first chip had 580 nets and 139 modules and measured approximately 11400× 11320λ. Channel definition produced 52 channels, 10 of which were empty. Running on a Symbolics 3600 Lisp Machine, 2PI spent roughly 90 minutes on channel routing, 90 seconds on crossing placement, 1 minute on global routing, and 30 seconds on channel definition.

The second chip had 412 nets and about 150 modules and measured approximately 6100×4300λ. Channel definition produced 41 nontrivial free channels, plus numerous tiny covered channels resulting from the power and ground wiring. 2PI spent roughly 1 hour on channel routing and 30 minutes on global routing. Running times for crossing placement and channel definition were comparable to those from the first chip. The increased time for global routing resulted from the fragmentation of covered channels.

With adjustments to handle new situations encountered in the two test chips, 2PI's new router succeeded on all but two free channels for each chip. The failed channels were too small to contain any successful routing. After a design engineer manually expanded the failed channels, 2PI successfully routed both chips.

Stenstrom and Mattheyses considered the proliferation of covered channels in the second chip a problem, even though this proliferation apparently did not prevent 2PI from laying out the chip.

13.4.3 Discussion

In hindsight, GE discovered that more than just "minor modifications" would be required to transform the prototype PI system into a useful production-quality system. Since, at this time, GE is not interested in undertaking a major rewrite of PI, GE has no immediate further plans for 2PI. Nevertheless, GE might abstract some of the ideas and algorithms from PI in future layout tools.

Through their experiences with PI, GE concluded that building an individual special-purpose layout tool is much easier than building a complete general-purpose VLSI layout system. Moreover, since most of GE's chips are very specialized, GE believes that special-purpose layout tools best suite their current needs.

Chapter 14

Open Problems

Organized in two sections, this chapter identifies and abstracts some of the important layout issues that arise in PI's placement algorithms. The first section precisely formulates three placement problems motivated by PI; the second section identifies some of the important issues that arise in mincut placement heuristics.

14.1 Abstractions of PI's Placement Problems

This section defines placement problems that distill in a simple and meaningful way some of the major considerations addressed by PI's placement algorithms. It would be useful to know which variations and restrictions of these problems are tractable and which are not.

14.1.1 A General Context for Placement Problems

We shall focus our attention on placement problems whose input consists of a set of *modules* to be placed, a set of *nets* which describe how the modules are to be interconnected, and a *target region* into which the modules are to be placed. Each net consists of a set of pins, and each pin is a point on some module.

We assume a layout methodology that lays out each chip in two separate phases: *placement* and *routing*. We further assume that there is a *layout cost* that measures the efficiency of a layout. For example, layout cost might be bounding area, wire length, or a linear combination of these measures. Similarly, we assume that there is an *estimated layout cost* that estimates how efficient the final layout of a placement will be. For example, estimated layout cost might estimate wire length by computing sum of the perimeters of the bounding boxes of the nets.

The goal of the placement problem is to find a placement of the modules in the target region that leaves enough room for routing while minimizing estimated layout cost. The goal of the routing problem is to lay down the wires in a way that minimizes

layout cost. Of course, we require that placements and layouts be *legal* in the sense that they satisfy the design rules for chip fabrication.

We say that *a placement is routable* if there is some routing of the placement. We also assume that there is a *routability measure* that estimates how "easy" it would be route the placement. For example, the routability measure might be based on a notion of layout density.

The placement problem involves a combination of geometric and graph theoretic concerns. At one extreme, if there are no nets and if the layout cost is bounding area, the placement problem becomes the bin packing problem [22]. At another extreme, if each module is a point, the placement problem becomes a graph embedding problem [381,380]. The PI System deals primarily with forms of the placement problem between these two extremes.

14.1.2 Three Placement Problems

To help understand what forms of the placement problem are efficiently solvable and what forms are not, we define three restricted placement problems motivated by PI. The first problem is a simple and general formulation of the placement problem; the second problem requires the placement to reflect a specified hierarchy; and the third problem deals with module orientation. Additional variations of these problems can be formed by restricting the positions of pins, limiting the number of pins on each net, and imposing different types of routability measures and layout costs.

Terminology

The following terms will be used in defining the restricted placement problems.

A *net over a set of modules* is a net each of whose pins lies on one of the modules.

A *placement tree over a set of modules* is a tree whose leaves are the modules. A placement tree is useful to express a hierarchy, such as a hierarchy that reflects the interconnectivity of the modules.

Let T be a placement tree and let P be a placement of T's modules. A *T-tree refinement of the placement P* is a modification of P that preserves the structure of T. The allowable modifications are recursively defined as follows. First, any module can be interchanged with any of its sibling modules in T. Second, for any subtree S of T, the placement of the leaves of S can be modified through any S-tree refinement. Tree refinements can be used to compute an exact placement from an approximate placement produced by a mincut heuristic.

An *orientation of a placement* is a modification of the placement achieved through flipping and rotating the modules around their centers, intended to improve the placement.

Unrestricted Uniform Placement

The *unrestricted uniform placement* problem is a simple formulation of a general placement problem. Modules have uniform size and may be centered on any point of a regular grid. Each pin is located at the center of its module. This problem is NP-complete.

Problem UUP. *Instance:* A set $\mathcal{M}$ of m modules, a set of n nets over $\mathcal{M}$, a wire length measure ϕ, and an integer d. *Restrictions:* Each module is a square; all modules have the same size; the target region is a regular square grid; each pin is located at its module center; and each module must be centered on a grid point. *Question:* Is there a legal placement P of the modules such that $\phi(P) \leq d$? *Note:* For the Manhattan wire length measure, this problem is NP-complete, by reduction from the mincut linear arrangement problem [22].

Uniform Tree Placement

Uniform tree placement is a restriction of the placement problem in which the placement must preserve a specified hierarchy. This problem can arise, for example, in mincut placement strategies that first compute a placement hierarchy. The complexity of this problem is not known.

Problem UTP. *Instance:* A set $\mathcal{M}$ of m modules, a set of n nets over $\mathcal{M}$, an initial placement P of $\mathcal{M}$, a placement tree T over $\mathcal{M}$, a wire length measure ϕ, and an integer d. *Restrictions:* The target region is a regular square grid; the placement tree is a complete binary tree; the initial placement is a slicing; each pin is located at its module center; and each module must be centered on a grid point. *Question:* Is there a legal, T-tree refinement τ of P such that $\phi(\tau(P)) \leq d$?

The slicing geometry of the initial placement makes dynamic programming a promising technique for solving this problem. Moreover, dynamic programming has been successfully applied to solve two similar problems. Stockmeyer considers a version of this problem in which the layout cost is bounding area and the orientation of each node in the placement tree is given, but in which modules may be arbitrary rectangles. Using dynamic programming, Stockmeyer solves his version of the problem in polynomial time [408]. As pointed out by Leiserson and Bhatt, the orientation information is not needed, but the tree must be height balanced in order for the Leiserson/Bhatt algorithm to run in polynomial time. Rivest and Fiduccia use dynamic programming to solve a separate one-dimensional version of the tree-placement problem in polynomial time [433]. Even if dynamic programming fails to solve the uniform tree placement problem, it would still be useful to know what placements

problems can be solved by dynamic programming.

Uniform Module Orientation

In the *uniform module orientation problem*, each module may be flipped and rotated, but must remain centered on its initial grid point. This problem can arise in the final adjustment of an approximate placement.

Problem UMO. *Instance:* A placement P of a set M of m modules and a set of n nets over M, a wire length measure ϕ, and an integer d. *Restrictions:* Each module is a square; all modules are the same size; the target region is a regular square grid; each pin is on a boundary of its module; and each module must be centered on a grid point. *Question:* Is there a legal orientation ρ of P such that $\phi(\rho(P)) \leq d$?

For the special case in which all modules lie along a line and the wire length measure is based on the Manhattan metric, a simple greedy strategy finds the optimal orientations. But the complexity of the general case of the uniform module orientation problem remains an open problem.

14.2 Mincut Issues

Motivated by PI's placement algorithms, the following informal questions identify several important issues that arise in using mincut strategies to place modules:

1. When applying mincut in a recursive fashion, what are the consequences of using various traversal orders (*e.g.* depth-first and breadth-first traversals)?

2. What are the consequences of applying mincut heuristics with different branching factors?

3. Koss suggests that, for each level in the mincut hierarchy, simultaneous partitioning of each rectangle at that level helps [392]. The intent of Koss's scheme is to enable mincut to make more of a global decision while refining each node the placement tree, thereby increasing the chance that mincut will find a globally optimal solution. What effect does this heuristic have?

4. When partitioning a circuit into two or more components, how should the components be balanced? What effects do various area-balance conditions have on the performance of mincut strategies? How can wire-area be incorporated into a balance condition?

5. In a mincut strategy which computes a slicing by recursively dividing a rectangular region, where should the cut line be drawn? How should be partition be oriented?

6. Once a mincut tree has been determined, how should an exact placement be calculated?

7. Would a multiple-pass mincut algorithm be useful? That is, could an initial mincut placement tree be useful to a mincut placement strategy?

8. How does the initial estimate of chip size affect the placement produced by a mincut procedure?

9. What preprocessing heuristics might be helpful? (For example, research by Thang Bui suggests that compressing sparse sections of graphs might help some mincut algorithms perform better [348].)

10. How can mincut strategies be applied to nonrectangular modules?

Chapter 15

Discussion

This chapter presents additional information and analysis about the PI System, including a brief review of related layout systems, a discussion of the implementation of the PI System, and a critique of the PI Project.

15.1 Related Work

This section briefly describes a few other layout systems that exist in addition to the PI System. This section also reviews some previous work on mincut placement techniques.

15.1.1 Other Layout Systems

To the best of my knowledge, the PI System is the first fully automatic placement and routing system for custom VLSI that has been described in the open literature. It is possible, however, that other automatic layout systems exist as propriety products in industry laboratories or as classified tools in government organizations. The rest of this section describes a few layout systems that were developed around the same time as the PI System.

Berkeley's *Magic* system is an intelligent, interactive, practical VLSI layout system [458]. Developed by John Ousterhout and his associates. Magic extends the capabilities of the earlier *Caesar* and *KIC2* layout editors [457,456]. Major features of Magic include a continuous design rule checker, routers that can operate in the presence of obstacles, and an interactive stretching and compacting operation known as *plowing*. As the name suggests, plowing rearranges a layout by pushing chip features out of the path of a "plow," while preserving design rules and electrical connectivity. To represent layouts, Magic maintains a set of logical *planes*, each of which is partitioned into rectangular *tiles*. Tile are "sewn" together by *corner stitches* that point to the orthogonal neighbors of the upper-right and lower-left tile corners.

Phoenix is an interactive program for placing modules on a custom chip. This program was designed and implemented at Stanford by former PI team member Chee-Seng Chow under the supervision of Brian Preas [397,460]. Phoenix is the placement phase of a layout process that performs placement and routing in two separate phases. Unlike PI, Phoenix makes no attempt to leave room for routing. Instead, Phoenix determines a *placement topology* which specifies the relative positions of the modules. Channels separating modules can be expanded without altering the placement topology. The routing phase intended for Phoenix will route each wire globally and then perform detailed channel routing channel by channel. Phoenix will process the channels in the order specified by a *channel order constraint graph*, which describes which channels can affect others. During detailed routing, Phoenix will expand channels as needed, but, when cycles in the channel order constraint graph exist, portions of the detailed routing process may have to be repeated until a solution is reached. Routing can alter module placement, but not the placement topology.

Using a combination of top-down and bottom-up approaches, Phoenix manipulates rectangular modules zero or more corners of which are missing. The top-down approach is based on a floor-planning mincut heuristic; the bottom-up approaches are based on a variety of heuristic search techniques. Phoenix also supports a few additional heuristics for improving placements. Although Phoenix was designed as a hierarchical system, currently Phoenix deals only with one "assembly" in the hierarchy.

Other interesting layout systems include Metalogic's *MetaSyn* silicon compiler [459] and Alberto Sangiovanni-Vincentelli's *Timberwolf* package, developed at Berkeley [461]. Timberwolf uses a simulated annealing heuristic [362,361,358] to place modules on a gate-array chip.

15.1.2 Previous Work on Mincut

Applying mincut strategies to solve placement problems has been tried since at least 1969, when, according to Leiserson, Günther used a mincut technique to place machines in a workshop [379,389]. Some researchers focus on the problem of placing uniform sized modules (*standard cells*). For example, Breuer studies how mincut schemes can place uniform sized modules on a regular grid [384], and Dunlop and Kernighan describe a mincut method for placing standard cells on a VLSI chip [387]. As do PI's placement algorithms, the Dunlop-Kernighan heuristic can make a local partition in the context of previously placed modules. The Dunlop-Kernighan heuristic operates as part of an interactive layout system developed at Bell Laboratories. Lauther applies a mincut strategy to place arbitrary sized rectangular modules on integrated circuits [393]. Lauther's method, which is part of a layout system developed at Siemens, improves an initial placement through three heuristics known as

rotation, squeezing, and reflection.

Graph partitioning plays an important role in mincut placement strategies. Although the basic graph partitioning problem is NP-complete [22], there are good heuristics for solving this problem. For example, Kernighan and Lin describe a general, iterative improvement heuristic for partitioning graphs [391]. Fiduccia and Mattheyses give a detailed implementation and adaptation of the Kernighan-Lin algorithm [388]. By exploiting a simple data structure, the Fiduccia-Mattheyses implementation runs in linear time per pass in the number of pins. As part of his research on the bisection size of random graphs, Thang Bui presents some graph partitioning heuristics [348]. Finally, as part of experimental work on simulated annealing, Johnson and his associates found that repeated trials of the Kernighan-Lin heuristic on random initial configurations usually obtained better results than a simulated annealing method run for a comparable amount of time[360].

15.2 PI System: Implementation, Contributors, Status, Documentation

The PI Project took place from about 1981 through 1983. During the first year, work focused on the slice router and global router. In the second year, the PI team designed of most of the other algorithms and finished its initial implementation of the signal routing phase. The third year was spent completing the design and implementation of the placement routines, power and ground routing algorithms, and resizer. This section presents more information about the PI System's implementation, contributors, current status, and documentation.

15.2.1 Implementation

The PI System is implemented in the Zetalisp dialect of Lisp and runs at MIT on a Symbolics 3600 Lisp Machine [468]. There are approximately 40,000 lines of source code (including comments), which were written by roughly fifteen people. PI completely placed and routed its first chip in November 1983.

Initial implementation of PI was done in Maclisp [469] on a PDP10 computer known as "ML". Several implementation difficulties arose from the PDP10's relatively small main memory. Eventually, the PI Project acquired a Symbolics 3600 Lisp Machine, at which time the PI team adopted ZetaLisp as the implementation language. For a while, the PI team attempted to keep the system Maclisp and Franzlisp compatible, but the team considered this effort too restrictive and eventually gave it up. Nevertheless, the team made a fairly careful attempt to isolate all machine and technology dependencies.

The decision to implement PI in Lisp was based on three reasons. First, Lisp's expressive power and object-oriented nature make it well-suited for developing prototype systems. Second, Lisp is the most popular and best supported language at MIT. Third, several existing VLSI tools (*e.g.* the DPL design system [341]) were already implemented in Lisp.

Three major considerations influenced the design of PI's algorithms—quality of results, implementation difficulty, and running time. The placement phase runs in time $O(mn \log_2 m)$ in the worst case, where m is the number of modules and n is the number of nets. But this formal time analysis means little since the worst-case inputs (*i.e.* complete graphs) are unrepresentative of typical inputs encountered in practice. In practice, the running time of PI is usually dominated by channel routing.

15.2.2 PI People

Numerous people contributed to the PI Project. Some people simply attended a few of the weekly meetings; others spent one or more years on the project. Although many of the algorithms were sketched out during the group meetings, each of the major contributors took primary responsibility for the design and implementation of one or more of the component algorithms. Most decisions regarding the structure of PI were made during the group meetings.

Given the large number of people who were involved with the PI Project, it is difficult to remember everyone's contributions. Nevertheless, I will now attempt to acknowledge the major participants. My apologies go to anyone whose contributions I may have omitted.

Ronald Rivest provided the inspiration, leadership, and continuity for the project, as well as many of the crucial ideas. He also designed and implemented the quick router, the Lee router, the initial version of the resizer, and many other parts of PI.

Alan Baratz designed and implemented the global router.

Arthur Chin designed and implemented the random example maker.

Chee-Seng Chow studied the problem of bottom-up pairing and implemented the geometric transformations used by the placement algorithms.

David Christman helped implement the slice router.

Jean Fitzmaurice worked on a routine to convert PI's output to SIF format.

Ali Ghaznavi created a few test examples.

Alain Hanover participated in several design discussions.

David Hsu wrote some of the graphics routines.

David Jilk implemented an early version of the pad placement algorithm.

Joe Kilian investigated ideas for building a hierarchical switch-box channel router.

Jim Koschella designed and implemented the slice router.

Michael Koss implemented an initial version of the hardening refinement.

Gordon Linoff designed and implemented the pad placement routine. He also wrote much of the graphics code.

Andrew Moulton designed and implemented the power-ground router.

Mark Novick analyzed the crossing-placement problem and designed a new crossing-ordering algorithm.

Ron Pinter contributed to the initial design of PI.

Flavio Rose helped maintain the entire system and wrote the *PI System User Manual.* He also revised the crossing placement routine.

Alan Sherman designed and implemented the placement heuristics. He also revised the initial implementation of the resizer.

Susmita Sur worked on the design and implementation of the resizer.

In addition, the following people also participated in the PI Project: Clark Baker, John Batali, Ramana Rao, Gerry Roylance, Alok Vijayvargia.

15.2.3 Current Status and Future Plans

All major pieces of the PI System have been implemented, except the resizer. The PI System runs at MIT and is available for use in laying out chips. A few students have used PI to lay out project chips, but there is still a very large overhead in using PI due to its software faults and untuned condition. The initial implementation of the PI System demonstrates the feasibility of PI's approach, but is not very useful as a practical layout tool.

A few extensions, alternate algorithms, and bells and whistles have not been implemented. In particular, the revised crossing-placement algorithm and the two additional placement strategies described in chapter 13 have not been implemented. Also, the fourth switch-box channel router planned for PI was not implemented, though a new greedy router was built at the General Electric Research Center.

PI has not been adequately tested. The major reason for this is that the initial implementation of the system is still in a rather rough state. Before it would make sense to benchmark the system, numerous software faults would have to be corrected and many of the algorithms would have to be fine-tuned.

To make PI into a practical layout tool, a complete reimplementation of the system would be required. Although no one has expressed an interest in undertaking such a revision, several researchers have expressed an interest in abstracting some of the ideas and algorithms from PI for use in other layout tools.[1]

The PI team has no plans for pursuing the PI Project any further, though several members of the team will likely continue working on problems motivated by PI.

[1] For example, Professor Richard Zippel has indicated an interest to incorporate some of PI's algorithms into his Schema system [470].

15.2.4 Works on the PI System

The most informative short description of the PI System appears as a 1982 conference paper by Rivest [452]. In his 1984 book on VLSI, Ullman also gives an overview of the PI System [345]. Both of these descriptions, though, are based on early designs of PI that have since changed in a few small ways.

In 1982, Flavio Rose wrote a user's manual for the PI System, but this document is now outdated [453].

Several theses and technical papers describe the PI System in detail. Alan Baratz's 1981 Ph.D. dissertation presents his work on global routing [443,449].

The Rivest-Fiduccia greedy router is explained in a 1982 conference paper [450], and Jim Koschella's adaptation of this router to PI is described in his 1981 Bachelor's thesis [444]. In a 1985 conference paper, Stenstrom and Mattheyses discuss the new greedy switch-box router that they developed for General Electric's 2PI project [429].

The Rivest-Fiduccia crossing-ordering algorithm appears in a 1983 unpublished paper [451]. Mark Novick's 1985 Bachelor's thesis further analyzes this algorithm and describes PI's new crossing-positioning heuristic [448].

A detailed description of PI's power and ground routines can be found in Andrew Moulton's 1984 Master's thesis [446,447].

15.3 Critique of the PI Project

This section takes an introspective look at the PI Project, identifying its major contributions, reflecting on its major design decisions, and evaluating how well it achieved its original objectives.

15.3.1 Major Contributions

The following five accomplishments stand out as the major contributions of the PI Project:

- A particular problem decomposition for laying out custom VLSI chips.
- Specialized component algorithms for solving subproblems that result from the problem decomposition.
- Identification of important issues and problems that arise in laying out VLSI chips.
- An initial implementation of the PI System.
- Practical experience in designing and building an automatic placement and routing system.

In addition, the PI Project helped stimulate interest in VLSI and provided a source of research problems for many students.

The single most important contribution of the PI Project is the method by which the PI System decomposes its layout problem. One novel feature of this problem decomposition is the unique crossing placement step, which plays a crucial role in reducing the signal routing task to independent fixed sized switch-box channel routing problems.

The PI System's component algorithms also contribute many useful ideas. These ideas come mainly through the goals, considerations, and strategies of the various component algorithms. For many of these algorithms, the goals, considerations, and strategies are more important than the particular details. But in heuristic algorithms, it is often more important what considerations are taken into account than how they are accounted for. Note also that the global routing algorithm and the new crossing placement algorithms come with proofs of several desirable properties [449,448].

For the placement algorithms, the major contribution is the framework in which these algorithms operate. This framework, which is centered around the placement tree, offers an effective means of supporting a variety of placement heuristics that combine geometric and graph-theoretic concerns.

Finally, the entire PI experience—including the initial implementation of the PI System—offers useful background to anyone planning to build an automatic layout system.

15.3.2 Reflections on the Major Design Decisions

This section takes a second look at each of the major design decisions that were made in the PI System.

As we knew at the beginning of the project, the decision to build a completely automatic layout system was an ambitious decision. Savings in labor costs, and the flexibility to make frequent changes in the design of a chip, make automatic systems extremely attractive. As the number of components on a chip increases, automatic layout systems will become a necessity to handle the huge number of modules and wires. In addition, the future holds promise for automatic layout systems producing higher quality layouts than those made by human experts. While it might be prudent for a company building a production-quality layout tool to tackle a less ambitious task, PI's goal of complete automation was a sound research direction.

Furthermore, PI's approach is relevant to nonautomatic, interactive systems. Although the PI team did not do so, it would be a straight-forward task to add many interactive features to the PI System. For example, the placement algorithms support certain types of partially specified placements. The global router, the quick router, and the Lee router can proceed with some nets already routed. However, some changes would need to be made to transform PI into an interactive system. In

particular, a more flexible data structure would be needed and many changes in the user interface would be required. Also, significant modifications would be required to enable the placement and detailed routing algorithms to lay out chips with arbitrary partial placements of wires and modules.

The decision to represent modules as arbitrarily shaped rectangles primarily affected the placement algorithms. PI's mincut approach works well with rectangular modules, especially since the mincut process produces a rectangular slicing of the logic box. Although PI's placement approach can be naturally adapted to handle other regular module shapes that tile the plane (*e.g.* triangles or hexagons), substantial modifications would be required to handle more complicated shapes such as arbitrary rectilinear modules. Of course, any arbitrary shape could be embedded in a bounding rectangle, but this strategy wastes lots of space. By contrast, the routing and resizing algorithms can easily accommodate more complicated shapes. For example, only the channel definition routine would have to be changed to enable the signal routing algorithms to handle arbitrary rectilinear modules.

The decision to decompose the layout process as a sequence of subproblems was an effective, practical way to proceed. PI's particular problem decomposition divided the layout process into natural subproblems of manageable size. This problem decomposition helped designers fine-tune parts of the layout process. Though, as we had known from the onset, PI can miss optimal solutions even if each subproblem is optimally solved.

PI's problem decomposition helps manage the complexity of the layout process in three ways. Separating the placement and routing phases divides the layout process by task. Distinguishing coarse routing from detailed routing divides the layout process by resolution. Breaking up the detailed routing problem into independent channel routing problems divides the layout process by locality.

The resizer plays a crucial role in PI's problem decomposition. All parts of PI are crucial to its success, but the resizer is especially important in guaranteeing that PI will eventually find a layout. Since the resizer expands and compresses the layout, the resizer can have a significant effect on the final placement of the modules. Also, since the resizer works in a symbolic world of constraints, the total behavior of the PI System has a certain symbolic flavor, even though the placement and routing algorithms work with absolute coordinates. Given the importance of the resizer, it may have been wiser to implement this component of the system first.

The PI team believed that high quality results could most easily be achieved through having PI work with absolute coordinates during routing. Since the resizer was never fully implemented, the effectiveness of PI's use of absolute coordinates remains difficult to assess.

PI's treatment of covered channels has a profound impact on signal routing. Since power and ground wires run all over the chip, they create numerous covered channels. Even more covered channels appear after the resizer widens the power and ground

wires to meet their current carrying requirements. This proliferation of covered channels causes PI to create more, but simpler free channels than it would without the covered channels. The final effect is to shift more burden on the global routing and crossing placement algorithms and less on the signal routers. Initially, the PI team underestimated the magnitude of this effect.

Although the crossing placement algorithm and some of the channel routers make some use of grids, PI does not require all chip features to be placed on a global grid. (Here, the term "global grid" refers to a regular grid that spans the entire chip, with spacings no smaller than the resolution of the manufacturing process.) PI treats corner points of chip features as objects, the representation of whose coordinates is left up to the computer running PI. But since the implementation of PI represents coordinates with finite precision, there is an implicit grid, albeit an extremely fine one.

Initially, the PI team viewed requiring that all chip features be placed on a global grid as an unnecessary and artificial restriction. The team did not want to impose restrictions on where pins can be located on modules, since that would impose an undo burden on module designers. However, the team did realize it would be possible to encase each module in a rectangle and route each pin to a nearby grid point on the edge of the encasing rectangle. More importantly, the team did not want to require that wires be routed on grid lines, since there are situations where that restriction makes it impossible to route an otherwise routable placement. The decision not to use a global grid gained some flexibility in routing, but caused several nuisances involving wire alignments, rounding errors, and the creation of tiny channels and wire segments.

When PI ran on the PDP10, it was a necessity to conserve space by not storing pointers to the neighbors of each wire rectangle. But this decision, together with the lack of a global grid, made it more difficult for the resizer to handle existing wires and contact cuts. Although there is no inherent reason why these difficulties could not be overcome, some of these difficulties were never adequately resolved. In hindsight, life would have been much simpler had PI maintained neighbor pointers and a global grid.

15.3.3 Conclusions

Overall, the PI Project was a success. The PI team designed and implemented an automatic layout system based on a sound problem decomposition and a large collection of layout heuristics. Chips laid out by PI demonstrate the feasibility and effectiveness of PI's approach.

The placement algorithms apply a top-down mincut strategy to place modules on a custom VLSI chip. The placement tree—the data structure around which these placement algorithms operate—provides an effective means for constructing heuris-

tics that combine geometric and graph-theoretic concerns.

The PI Project falls roughly into the area of advanced research and development. Emphasis on algorithm design gave the project a theoretical flavor, yet the implementation of the PI System presented a large practical task.

It is difficult to evaluate projects like the PI Project. Theoreticians look for theorems; practioners want to see experimental results. But often theorems apply only to special cases or to asymptotic behaviors, and experimental results tend to say more about the quality of the implementation than about the underlying ideas. Moreover, heuristics are difficult to analyze and the assessment of the PI System is complicated by interactions among the algorithms, user interface, and implementation.

At times, the PI Project was caught in between theory and practice. Work on algorithm design and analysis took away from the implementation, and the implementation siphoned resources away from design. Although the emphasis was always on algorithm design, implementation sometimes forced the designers to make difficult engineering choices. I believe that both the theoretical and practical dimensions of the project would have benefited from a cleaner separation of design and implementation. But given the ambitious character of the project, the PI System would never have been completed if implementation had waited until all algorithms were perfected to the complete satisfaction of the designers. Moreover, the implementation gave the designers useful feedback and helped ensure that no important detail was overlooked.

Although there was never any attempt at writing production-quality software, the initial implementation of the PI System fell short of one of its goals to be useful to students in laying out project chips. While a rewrite of the PI System would likely succeed at this goal, the current software is not very useful.

Many factors affected the quality of PI's software. First, there was never a full-time programmer nor a primary person in charge of the software effort. Too many people wrote software and many of the people were with the project for a relatively short period of time. Moreover, most of the participants had very little software experience and were not interested in writing production-quality software. Initial work on the PDP10 was hindered by a poor programming environment, but, even after work switched to a Symbolics 3600 Lisp Machine, the PI team never made full use of that machine's features. The lack of a global grid created some systemic complications, and specifications of the PI System evolved throughout the project. On the positive side, PI's problem decomposition helped break up the software task into fairly independent modules. Independent of its VLSI connections, the PI project provides an interesting case study in software engineering.

High-quality software development is expensive and almost always requires at least one major revision of the entire system. To become a useful production-quality layout system, the PI system needs to be rewritten. I estimate this could be done with three full-time programmers working over one year. But what is more likely to happen is for other researchers and developers to extract the major algorithms and

ideas from the PI Project and adapt them into new systems.

Bibliography

This detailed bibliography lists and organizes sources I used during my dissertation research. The bibliography is organized hierarchically by subject categories and attempts to give clear pointers to sources referred to in the body of the dissertation. In addition, the bibliography strives to list each technical work that may have indirectly influenced my research, even if the work is not referenced in the body of the thesis. But no attempt was made to make the bibliography a comprehensive guide to literature in VLSI and cryptology.

The bibliography is divided into three major sections: basic mathematics and computer science, cryptology, and VLSI. Each of these sections is subdivided into additional subject categories. Within each subject category, works are listed alphabetically by the last name of the first author. Each work is assigned a primary subject category. Of course, the choice of categories and the assignment of works to these categories is somewhat arbitrary, and for some works several subject categories could have been assigned.

For ease of finding entries in the bibliography, each work is numbered by the order in which the work's primary citation appears. A single numbering scheme spans the entire bibliography. As a supplementary indexing mechanism, an outline of subject categories begins on page 180.

Outline of Bibliography

General Works on Mathematics and Computer Science

Number Theory and Algebra

[1] Albert, Adrian A., *Fundamental Concepts of Higher Algebra*, University of Chicago Press (Chicago 1966).

[2] Carmichael, Robert D., *Introduction to the Theory of Groups of Finite Order*, Dover (New York, 1956).

[3] Hardy, G. H.; and E. M. Wright, *An Introduction to the Theory of Numbers*, Oxford Univ. Press (London, 1971).

[4] Herstein, I. N., *Topics in Algebra*, Blaisdell (New York, 1975).

[5] Hungerford, Thomas W., *Algebra*, Springer (New York, 1974).

[6] LeVeque, *Fundamentals of Number Theory*, Addison-Wesley (1977).

[7] Niven, Ivan; and Herbert S. Zuckerman, *An Introduction to the Theory of Numbers*, John Wiley (New York, 1980).

[8] Rotman, Joseph J., *The Theory of Groups: An Introduction*, Allyn and Bacon (Boston, 1978).

[9] Wielandt, Helmut, *Finite Permutation Groups*, Academic Press (New York, 1964).

Probability and Statistics

[10] Cramer, H., *Mathematical Methods of Statistics*, Princeton Univ. Press (1974).

[11] Feller, W., *An Introduction to Probability Theory and its Applications*, vols. I–II, John Wiley (New York, 1957).

[12] Good, Irving John, *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*, MIT Press (1965).

[13] Larsen, Richard J; and Morris L. Marx, *An Introduction to Mathematical Statistics and its Applications*, Prentice-Hall (Englewood Cliffs, N.J., 1981).

[14] Osteyee, David Bridston; and Irving John Good, *Information, Weight of Evidence, the Singularity between Probability Measures and Signal Detection*, Springer (Berlin, 1974).

Combinatorics

[15] Comet, Louis, *Advanced Combinatorics*, D. Reidel Publishing Company (1974).

[16] Lovász, L., *Combinatorial Problems and Exercises*, North-Holland (Amsterdam, 1979).

[17] Roberts, Fred, *Applied Combinatorics*, Prentice-Hall (1984).

Algorithms and Complexity Theory

[18] Aho, Alfred; John E. Hopcroft; and Jeffrey Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley (Reading, MA, 1974).

[19] Berge, C., *The Theory of Graphs and its Applications*, John Wiley (New York, 1966).

[20] Cook, S. A., "The complexity of theorem-proving procedures," *Proceedings of the 3rd STOC* (1971), 151–158.

[21] Even, Shimmon, *Graph Algorithms*, Computer Science Press (Potomac, MD, 1979).

[22] Garey, Michael R.; and David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman (San Francisco, 1979).

[23] Hillier, Frederick S.; and Gerald J. Lieberman, *Operations Research*, Holden-Day (San Francisco, 1974).

[24] Hopcroft, John E.; and Jeffrey Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley (Reading, MA, 1979).

[25] Karp, Richard M., "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, eds., Plenum Press (1972), 85–103.

[26] Knuth, Donald E., *Fundamental Algorithms* in *The Art of Computer Programming*, vol. I, Addison-Wesley (Reading, MA, 1973).

[27] Knuth, Donald E., *Seminumerical Algorithms* in *The Art of Computer Programming*, vol. II, Addison-Wesley (Reading, MA, 1973).

[28] Knuth, Donald E., *Sorting and Searching* in *The Art of Computer Programming*, vol. III, Addison-Wesley (Reading, MA, 1973).

[29] Kolmogorov, A., "Three approaches to the quantitative definition of information," *Problems of Information Transmission*, **1** (January–March, 1965), 1–7.

[30] Levin, L. A., "Universal sorting problems," *Problems of Information Transmission*, **9** (1973), 265–266.

[31] Levin, L. A., "Problems complete in 'average' instance," *Proceedings of the 16th STOC* (April 1984), 465.

[32] Lewis, Harry R.; and Christos H. Papadimitriou, "The efficiency of algorithms," *Scientific American*, **238** (January 1978), 96–109.

[33] Papadimitriou, Christos H.; and Kenneth Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall (1982).

[34] Purdom, Paul W. Jr.; and Cynthia A. Brown, *The Analysis of Algorithms*, Holt, Rinehart, and Winston (New York, 1985).

[35] Savage, John E., *The Complexity of Computing*, John Wiley (New York, 1978).

Information Theory

[36] Gallager, Robert G., *Information Theory and Reliable Communication*, John Wiley (New York, 1968).

[37] Kullbach, Solomon, *Information Theory and Statistics*, John Wiley (New York, 1959).

[38] Shannon, Claude E., "The mathematical theory of communication," *Bell System technical Journal* (July and October, 1948), 656–715.

Works on Cryptology

Basic Sources

Bibliographies

[39] Diffie, Whitfield, "A technical bibliography of cryptography," technical paper, BNR Inc., Mountain View, CA (October 1983).

[40] Diffie, Whitfield, Results of an unclassified computerized literature search in cryptology using the Defense Technical Information Center, unpublished document (1984).

[41] Levine, Jack, *United States Cryptographic Patents 1861-1981*, Cryptologia (Terre Haute, Indiana, February 1983).

[42] Price, Wyn L., Annotated bibliographies of recent publications on data security and cryptography, National Physical Laboratory technical reports, Teddington, England (January 1978 to date).

[43] Sherman, Alan T., Results of a computerized literature search in cryptology using CONIT and the Science Abstracts, unpublished document, MIT Laboratory for Computer Science (May 1984).

[44] Schulman, David, *An Annotated Bibliography of Cryptography*, Garland Publishing (New York, 1976).

See also the bibliographies in [58,60,62,69].

Major Annual Conferences for Technical Cryptology

- Crypto, A workshop on the theory and application of cryptographic techniques, sponsored by the International Association for Cryptologic Research, Univ. of Cal. Santa Barbara (1981 to date).

[45] Gersho, Allen, ed., "Advances in cryptology: a report on Crypto 81," ECE report 82–04, Department of EECS, U. C. Santa Barbara (1982).

[46] Chaum, David; Ronald Rivest; and Alan T. Sherman, eds., *Advances in Cryptology: Proceedings of Crypto 82*, Plenum Press (New York, 1983).

[47] Chaum, David, ed., *Advances in Cryptology: Proceedings of Crypto 83*, Plenum Press (New York 1984).

[48] Blakley, G. R.; and David Chaum, eds., *Advances in Cryptology: Proceedings of Crypto 84*, Springer (1985).

[49] Williams, H. C., ed., *Advances in Cryptology: Proceedings of Crypto 85*, Springer (1986).

• Eurocrypt, sponsored by the International Association for Cryptologic Research (1982 to date).

[50] Beth, Thomas, ed., *Cryptography, Proceedings of the Workshop on Cryptography, Burg Feuerstein, Germany, March 29–April 2, 1982*, Springer (Berlin, 1983).

[51] Beth, Thomas; N. Cot; and I. Ingemarsson, eds., *Advances in Cryptology: Proceedings of Eurocrypt 84*, Springer (1984).

[52] Pichler, Franz, ed., *Advances in Cryptology—Eurocrypt 85*, Springer (1985).

• *IEEE Symposium on Security and Privacy (SSP)*, sponsored by the IEEE Technical Committee on Security and Privacy (1979 date). Proceedings published by IEEE Computer Society Press.

Other Conferences

[53] Fåk, Viiveke, ed., *Security, IFIP/Security '83: Proceedings of the First Security Conference, Stockholm, Sweden, 16–19 May, 1983*, North-Holland (Amsterdam, 1983).

• Workshop on the Mathematical Aspects of Security, MIT Endicott House, sponsored by the National Science Foundation (Dedham, MA, June 1985).

• *Annual Symposium on Foundations of Computer Science (FOCS)*, sponsored by the IEEE Computer Society's Technical Committee on Mathematical Foundations of Computing (1960 to date). Proceedings published by the IEEE Computer Society Press.

• *International Colloquium on Automata, Languages, and Programming (ICALP)*, sponsored by European Association for Theoretical Computer Science (1974 to date). Proceedings published by Springer.

• *Annual ACM Symposium on Theory of Computing (STOC)*, sponsored by the ACM Special Interest Group for Automata and Computability (SIGACT), (1969 to date). Proceedings published by ACM.

Collections

[54] Davies, Donald, ed., *Tutorial: The Security of Data in Networks*, IEEE Computer Society Press (1982).

[55] DeMillo, Richard A.; David P. Dobkin; Anita K. Jones; and Richard J. Lipton, *Foundations of Secure Computations*, Academic Press (New York, 1978).

[56] Longo, G., ed., *Secure Digital Communications*, Springer (Vienna 1983).

[57] Simmons, Gustavus J., ed., *Secure Communications and Asymmetric Cryptosystems*, AAAS Selected Symposium, Westview Press (Boulder Colorado, 1982).

Survey Works

[58] Beker, Henry.; and Fred Piper, *Cipher Systems: The Protection of Communications*, John Wiley (New York, 1982).

[59] Davies, Donald W.; and W. L. Price, *Security for Computer Networks: An Introduction to Data Security in Teleprocessing and Electronic Funds Transfer*, John Wiley (New York, 1984).

[60] Denning, Dorothy E. R., *Cryptography and Data Security*, Addison-Wesley (Reading, MA, 1982).

[61] Denning, D.; and P. Denning, "Data security," *ACM Computing Surveys*, **11** (September 1979), 227–249.

[62] Diffie, Whitfield; and Martin E. Hellman, "Privacy and authentication: An introduction to cryptography," *Proceedings of the IEEE*, **67** (March 1979), 397–427.

[63] Diffie, Whitfield, "Security problems in modern communications," working paper (August 1984).

[64] Feistel, Horst, "Cryptography and Computer Privacy," *Scientific American*, **228** (May 1973), 15–23.

[65] Gaines, Helen Fouché, *Cryptanalysis: A Study of Ciphers and Their Solution*, Dover (1956).

[66] Konheim, Alan G., *Cryptography: A Primer*, John Wiley (New York, 1981).

[67] Lempel, Abraham, "Cryptology in transition: a survey," *ACM Computing Surveys*, **11** (December 1979), 285–303.

[68] Meyer, Carl H.; and Stephen M. Matyas, *Cryptology: A New Dimension in Computer Data Security*, John Wiley (New York, 1982).

[69] Rivest, Ronald L., "Cryptology," in *Handbook of Theoretical Computer Science*, Albert Meyer, ed., North-Holland (1986), to appear.

[70] Simmons, Gustavus J., "Symmetric and asymmetric encryption," *Computing Surveys*, 2 (December 1979), 305–330.

History and Policy

History

[71] Deavours, Cipher; and Louis Kruh, *Machine Cryptography and Modern Cryptanalysis*, Artech House (Dedham, MA, 1985).

[72] Bamford, James, *The Puzzle Palace: A Report on NSA, America's most Secret Agency*, Houghton Mifflin (Boston, 1982).

[73] Kahn, David, "Cryptology and the origins of spread spectrum," *IEEE Spectrum* (September 1984), 70–80.

[74] Kahn, David, *Kahn on Codes*, Macmillan (New York, 1983).

[75] Kahn, David, *The Codebreakers, the Story of Secret Writing*, Macmillan (New York, 1967).

[76] Kozaczuk, Wladyslav, *Enigma: How the German Machine Cipher was Broken, and How it was Read by the Allies in World War Two*, edited and translated by Christopher Kasparek, University Publications of America (1984).

[77] Parrish, T., *The Ultra Americans: The U.S. Role in Breaking Nazi Codes*, Stein and Day (Briarcliff Manor, NY, 1986).

[78] Randell, Brian, "The COLOSSUS," technical report 90, Univ. of Newcastle upon Tyne (June 1976).

[79] Welschman, Gordon, *The Hut Six Story, Breaking the Enigma Codes*, McGraw-Hill (New York, 1982).

[80] Winterbotham, F. W., *The Ultra Secret*, Futura (London, 1975).

[81] Yardly, Herbert O., *The American Black Chamber*, Bobbs-Merrill (Indianapolis, 1931).

Policy

[82] Kolata, Gina, "Codes go public," *Boston Globe* (September 30, 1985), 44.

[83] Inman, Bobby R., "The NSA perspective on telecommunications protection in the nongovernmental sector," *Signal*, **33** (March 1979), 6–79.

[84] Sanders, Sylvia, "Data privacy: what Washington doesn't want you to know," *Reason* (January 1981), 24–37.

[85] Shapley, Deborah; and Gina Bari Kolata, "Cryptology: scientists puzzle over threat to open research," *Science*, **209** (September 1977).

Specialized Topics in Mathematics and Theoretical Computer Science

Random Functions and Random Permutations

[86] Bovey, John; and Alan Williamson, "The probability of generating the symmetric group," *Bull. London Math Society*, **10** (1978), 91–96.

[87] Bovey, J. D., "An approximate probability distribution for the order of elements of the symmetric group," *Bull. London Math Society*, **12** (1980), 41–46.

[88] Bovey, J. D., "The probability that some power of a permutation has small degree," *Bull. London Math Society*, **12** (1980), 47–51.

[89] Dixon, John D., "The probability of generating the symmetric group," *Math Zentrum*, **110** (1969), 199–205.

[90] Erdõs, P.; and P. Turán, "On some problems of a statistical group-theory. II," *Acta Mathematica Academiae Scientiarum Hungaricae*, vol. 18, nos. 1–2 (1967), 151–163.

[91] Goldreich, O.; S. Goldwasser; and S. Micali, "How to construct random functions," *Proceedings of the 25th FOCS* (1984), 464–479.

[92] Harris, Bernard, "Probability distributions related to random mappings," *Annals of Math. Statistics*, **31** (1959), 1045–1062.

[93] Purdom, Paul W.; and J. H. Williams, "Cycle length in a random function," *Transactions of the American Mathematics Society*, **133** (1968), 547–551.

[94] Shepp, L. A.; and S. P. Lloyd, "Ordered cycle lengths in a random permutation," *Transactions of the American Mathematics Society*, (February 1966), 340–357.

Cycle Detection

[95] Allender, Eric; and Maria Klawa, "Improved Lower Bounds for the Cycle Detection Problem," working paper.

[96] Fich, Faith E., "Lower bounds for the cycle detection problem," *Proceedings of the 13th STOC* (1981), 96–105.

[97] Hinsdale, John Kelley, "Implementing the Sedgewick-Szymanski cycle detection algorithm," BS Thesis, Dept. of Electrical Engineering and Computer Science, MIT (February 1985).

[98] Sedgewick, Robert, and Thomas G. Szymanski, "The complexity of finding periods," *Proceedings of the 11th STOC*, (1979), 74–80.

[99] Sedgewick, Robert; Thomas G. Szymanski; and Andrew C. Yao, "The complexity of finding cycles in periodic functions," *Siam Journal on Computing*, **11** (1982), 376–390.

Factoring

[100] Bach, Eric, "How to generate random integers with known factorization," *Proceedings of the 15th STOC* (April 1983), 184–192.

[101] Berlekamp, Elwyn R., "Factoring polynomials over large finite fields," *Math. Comp.*, **24** (1970), 713–735.

[102] Brent, Richard P., "Analysis of some new cycle-finding and factorization algorithms," technical report, Department of Computer Science, Australian National University (1979).

[103] Davis, James, A.; Diane Holdbridge; and Gustavus Simmons, "Status report on factoring" in [48].

[104] Lenstra, A.; H. Lenstra; and L. Lovász, "Factoring polynomials with rational coefficients," *Mathematische Ann.*, **261** (1982), 513–534.

[105] Miller, Victor, "Lenstra's factoring algorithm," (1985).

[106] Pollard, J. M., "A Monte Carlo method for factorization," *Bit*, **15** (1975), 331–334.

[107] Pomerance, Carl, "Analysis and comparison of some integer factoring algorithms," in *Computational Methods in Number Theory*, H. W. Lenstra and R. Tijdeman, eds., Math. Centrum Tract 154 (Amsterdam 1982), 89–139.

Primality Testing

[108] Adleman, Leonard M.; and M. A. Huang, "Recognizing primes in random polynomial time," unpublished abstract (1986).

[109] Adleman, L.; C. Pomerance; and R. Rumely, "On distinguishing prime numbers from composite numbers," *Annals of Math.* **117** (1983), 173–206.

[110] Goldwasser, Shafi; and Joe Kilian, "Almost all primes can be quickly certified," *Proceedings of 18th STOC* (May 1986), 316–329.

[111] Rabin, M., "Probabilistic algorithms for testing primality," *Journal of Number Theory*, **12** (1980), 128–138.

[112] Solovay R.; and V. Strassen, "A fast Monte-Carlo test for primality," *SIAM Journal on Computing*, **6** (1977), 84–85.

Discrete Logarithm

[113] Adleman, Leonard M., "A subexponential algorithm for the discrete logarithm problem with applications to cryptography," *Proceedings of the 20th FOCS* (1979), 55–60.

[114] Coppersmith, D., "Evaluating Logarithms in $GF(2^n)$," *Proceedings of 16th STOC* (1984), 201–207.

[115] Odlyzko, A. M., "Discrete logarithms in finite fields and their cryptographic significance" in [51], 224–314.

[116] Pohlig, S.; and M. Hellman, An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance," *IEEE Trans. on Information Theory*, **IT-24** (January 1978), 106–110.

Information-Lossless Computation

[117] Bennett, Charles H.; and Rolf Landauer, "The Fundamental Physical Limits of Computation," *Scientific American*, **253** (July 1985), 48–56.

[118] Bennett, Charles H., "Logical reversibility of computation," *IBM Journal of Research and Development*, **17** (November 1973), 525–532.

[119] Kurmit, A. A., *Information-Lossless Automata of Finite Order*, John Wiley (New York, 1974).

[120] Toffoli, Tommaso, "Reversible computing," technical memorandum TM–151, MIT Lab. for Computer Science (February 1980).

Other Specialized Works in Mathematics

[121] Adleman, L.; K. manders; and G. Miller, "On taking roots in finite fields," *Proceedings of 18th FOCS* (1977), 175–177.

[122] Berlekamp, Elwyn R., *Algebraic Coding Theory*, McGraw-Hill (1968).

[123] Berlekamp, E. R.; R. J. McEliece; and H. van Tilborg, "On the inherent intractability of certain coding problems," *IEEE Trans. on Info. Theory*, **IT–24** (1978), 384–386.

[124] Angluin, Dana, "Lecture notes on the complexity of some problems in number theory," technical report 243, Dept. of Computer Science, Yale Univ. (August 1982).

[125] Chandra, Ashok K., "Efficient compilation of linear recursive programs," technical report no. STAN–CS–72–282, Computer Science Dept., Stanford Univ (April 1972).

[126] Reif, John, "Probabilistic algorithms in group theory," technical report, Aiken Computation Laboratory, Harvard Univ. (1984).

[127] Sattler, J.; and C. P. Schnorr, "Generating random walks in groups," unpublished manuscript (October 1983).

Theory of Cryptology and Cryptographic Security

Complexity Theory and Cryptographic Security

[128] Adleman, Leonard M, "On new foundations for cryptology," unpublished manuscript, MIT Lab. for Computer Science (1980).

[129] Angluin, D.; and D. Lichtenstein, "Provable security of cryptosystems: a survey," technical report TR–288, Dept. of Computer Science, Yale Univ. (October 1983).

[130] Blum, Manuel; and Silvio Micali, "How to generate cryptographically strong sequences of pseudo random bits," *SIAM Journal on Computing*, **13** (November 1984), 850–864.

[131] Boyack, Stephen W., "The robustness of combinatorial measures of boolean matrix complexity," Ph.D. thesis, Dept. of Mathematics, MIT (June 1985).

[132] Brassard, Gilles, "Relativized cryptography," *Proceedings of the 20th FOCS* (1979), 383–391.

[133] Brassard, Gilles, "A time-luck tradeoff in cryptology," *JCSS*, **22** (June 1981), 280–311.

[134] Brassard, Gilles, "A note on the complexity of cryptography," *IEEE Transactions on Information Theory*, **IT–25** (March 1979), 232–233.

[135] Even, Shimon; and Y. Yacobi, "An observation concerning the complexity of problems with few solutions and its application to cryptography," technical report TR–167, Computer Science Dept., Technion (1980).

[136] Even, Shimon; and Y. Yacobi, "Cryptocomplexity and NP-completeness," technical report 172, Computer Science Dept., Technion, Israel (March 1980).

[137] Even, Shimon; Alan L Selman; and Yacov Yacobi, "The complexity of promise problems with applications to public-key cryptography," *Proceedings of ICALP82* (1982), 502–509.

[138] Goldwasser, Shafi; and Silvio Micali, "Probabilistic encryption," *JCSS* **28** (1984), 270–299.

[139] Goldwasser, S.; S. Micali; and P. Tong, "Why and how to establish a private code on a public network," *Proceedings of the 23rd FOCS* (1982), 134–144.

[140] Grollman, J.; and A. L. Selman, "Complexity measures for public-key cryptosystems," *Proceedings of the 25th FOCS* (1984), 495–515.

[141] Halsey, J. C., "Finite automata admitting inverses with some applications to cryptography," Ph.D. Thesis, Dept. of Mathematics, North Carolina State University (1970).

[142] Levin, L., "One-way functions and pseudorandom generators," *Proceedings of the 17th STOC* (1985), 363–365.

[143] Lieberherr, Karl, "Uniform complexity and digital signatures," *Theoretical Computer Science*, **16** (October 1981), 99–110.

[144] Micali, Silvio; Charles Rackoff; and Robert Sloan, "The notion of security for probabilistic cryptosystems," unpublished manuscript (1986).

[145] Selman, Alan L., "Complexity measures for public-key cryptosystems," *Proceedings of the 25th FOCS* (1984), to appear.

[146] Selman, Alan L., "Remarks about natural self-reducible sets in NP and complexity measures for public-key cryptosystems," submitted to *ICALP84* (1984).

[147] Sloan, Robert H., "The notion of security for probabilistic public-key cryptosystems," master's thesis, MIT Dept. of EECS (1986).

[148] Yao, A., "Theory and applications of one-way functions," *Proceedings of the 23rd FOCS* (1982), 80–91.

Information Theory and Cryptographic Security

[149] Hellman, Martin E., "An extension of the Shannon theory approach to cryptography," *IEEE Transactions on Information Theory*, **IT–23** (May 1977), 289–294.

[150] Shannon, Claude E., "Communication theory of secrecy systems," *Bell System Technical Journal*, **28** (October 1949), 656–715.

[151] Sloane, N. J. A., "Error-correcting codes and cryptography" in *The Mathematical Gardner*, D. Klarner, ed., Wadsworth (Belmont, CA, 1981), 346–382.

[152] Wyner, Aaron D., "The wire tap channel," *Bell System Technical Journal*, **54** (October 1975), 1355–1387.

Physics and Cryptographic Security

[153] Bennett, Charles H.; Gilles Brassard; Seth Breidbart; and Stephen Wiesner, "Quantum cryptography, or unforgeable subway tokens," in [46], 267–275.

Public-Key Cryptography

[154] Diffie, Whitfield; and Martin E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, **IT-22** (November 1976), 644–654.

[155] Hellman, M., "The mathematics of public-key cryptography," *Scientific American* (February 1979), 146–157.

[156] Merkle, Ralph C., "Secure communications over insecure channels," *CACM*, **21** (April 1978), 294–299.

Specific Cryptosystems

Federal Standards Involving DES

[157] "Data Encryption Standard," National Bureau of Standards, Federal Information Processing Standards Publications No. 46 (January 15, 1977).

[158] "DES modes of operations," Federal Information Standards Publication No. 81 (December 1980).

[159] "Guidelines for implementing and using the NBS Data Encryption Standard," National Bureau of Standards, Federal Information Processing Standards Publications No. 74 (April 1981).

[160] "Telecommunications: Interoperability and security requirements for use of the Data Encryption Standard in the physical layer of data communication," General Service Administration, FS 1026 (August 3, 1983).

Data Encryption Standard

[161] Branstad, D. K.; Gait, J.; and S. Katzke, "Report of the workshop on cryptography in support of computer security," National Bureau of Standards Report NBSIR 77–1291 (September 21-22, 1976).

[162] Brickell, E. F.; J. H. Moore; and M. R. Purtill, "Structure in the S-boxes of the DES," extended abstract, presented at Crypto 86 (August 1986).

[163] Chaum, David, "Breaking 4,5,6 and more rounds of DES: An interm report on investigation of PC-2" in [49].

[164] Coppersmith, Don; and Edna Grossman, "Generators for certain alternating groups with applications to cryptology," *Siam Journal on Applied Mathematics*, 29 (December 1975), 624–627.

[165] Davies, Donald W., "Some regular properties of the DES" in [46], 89–96.

[166] Davies, Donald W.; and G. I. P. Parkin, "The average size of the key stream in output feedback encipherment" in [50], 263–279.

[167] Davies, Donald W.; and G. I. P. Parkin, "The average size of the key stream in output feedback mode" in [46], 97–98.

[168] Davio, Marc, *et al.*, "Analytical Characteristics of the DES," in [47] (1983).

[169] Davio, Mark; Yvo Desmedt; and Jean-Jacques Quisquater, "Dependence of output on input in DES: small avalanche characteristics" in [48].

[170] Davio, Mark; Yvo Desmedt; Jozef Goubert; Frank Hoornaert; and Jean-Jacques Quisquater, "Efficient hardware and software implementations for the DES" in [48].

[171] Davis, R. M., "The Data Encryption Standard in perspective," *Computer Security and the Data Encryption Standard*, National Bureau of Standards Special Publication 500–27 (February 1978).

[172] Desmedt, Yvo, "Analysis of the security and new algorithms for modern industrial cryptography," dissertation, Department Elektrotechniek, Katholieke Universiteit Leuven (October 1984).

[173] Diffie, Whitfield; and Martin E. Hellman, "Exhaustive cryptanalysis of the NBS Data Encryption Standard," *Computer*, **10** (June 1977), 74–84.

[174] Feldman, Frank, "A new spectrum test for nonrandomness and the DES," working paper (April 1985).

[175] Gait, Jason, "A new nonlinear pseudorandom number generator," *IEEE Transactions on Software Engineering*, **SE-3** (September 1977), 359–363.

[176] Goldreich, Oded, "DES-like functions can generate the alternating group," *IEEE Transactions on Information Theory*, **IT–29** (1983), 863–865.

[177] Gordon, J. A.; and H. Retkin, K., "Are big S-boxes best?" in [50].

[178] Grossman, Edna; and Bryant Tuckerman, "Analysis of a Feistel-like cipher weakened by having no rotating key," IBM research report RC 6375 (#27489), (January 31, 1977).

[179] Grossman, Edna; and Don Coppersmith, "Generators for certain alternating groups with applications to cryptology," IBM technical report RC 4741 (February 26, 1974).

[180] Hellman, Martin E., *et al.*, "Results of an initial attempt to cryptanalyze the NBS Data Encryption Standard," technical report SEL 76–042, Information Systems Laboratory, Stanford Univ. (November 1976).

[181] Hellman, Martin E., "A cryptanalytic time-memory tradeoff," technical report, Stanford Univ. (1978).

[182] Hellman, Martin E., "DES will be totally insecure within ten years," *IEEE Spectrum*, **16** (July 1979).

[183] Hellman, Martin E.; and Justin M. Reyneri, "Distribution of Drainage in the DES," in [46] (1982), 129–131.

[184] Jueneman, Robert R., "Analysis of certain aspects of output-feedback mode," in [46] (1982), 99–127.

[185] Kaliski, Burton S.; Ronald L. Rivest; and Alan T. Sherman, "Is the Data Encryption Standard a Group?" in [52], 81–95.

[186] Kaliski, Burton S.; Ronald L. Rivest; and Alan T. Sherman, "Is DES a pure cipher? (Results of more cycling experiments on DES)" in [49].

[187] Kaliski, Burton S., "On the design of fast cycle detection hardware for DES," master's thesis, Dept. of EECS, MIT, to appear.

[188] Kolata, G. B., "Computer encryption and the National Security Agency," *Science*, **197** (July 29, 1977), 438–440.

[189] Kolata, Gina, "Flaws found in popular code," *Science*, **28** (January 1983), 369–370.

[190] Meissner, P., ed., "Report of the workshop on estimation of significant advances in computer technology," National Bureau of Standards Report NBSIR 76–1189 (December 1976).

[191] Momirov, Milan, "LSI implementation of the Data Encryption Standard," *Computer Design* (June 1980), 158–164.

[192] Moore, J. H.; and G. J. Simmons, "Cycle structure of the DES with weak and semiweak keys," extended abstract, presented at Crypto 86 (August 1986).

[193] Morris, R.; N. J. A. Sloan; and A. D. Wyner, "Assessment of the NBS proposed Data Encryption Standard," *Cryptologia*, **1** (July 1977), 281–291.

[194] Reeds, J. A.; and J. L. Manferdell, "DES has no per round linear factors," in [48].

[195] Schaumüller-Bichl, Ingrid, "Zur Analyse des Data Encryption Standard und Synthese verwandter Chiffiersysteme," dissertation, Johannes-Kepler University, Linz (May 1981).

[196] Shamir, Adi, "Is the DES secure?" in [49].

[197] Tuchman, W. L., talk presented at National Computer Conference, (June 1978).

[198] Tuchman, W. L., "Hellman presents no shortcuts to the DES," *IEEE Spectrum*, **16** (July 1979), 40–41.

[199] "Unclassified summary: Involvement of NSA in the development of the Data Encryption Standard," staff report of the Senate Select Committee on Intelligence, United States Senate (April 1978).

Knapsack Cryptosystems

[200] Adleman, Leonard, "On breaking iterated knapsacks" in [46], 303–308.

[201] Adleman, Leonard, "On breaking generalized knapsack public-key cryptosystems," *Proceedings of the 15th STOC* (1983), 402–412.

[202] Brickell, Ernest F.; and Gustavus J. Simmons, "A status report on knapsack based public-key cryptosystems," *Congressus Numerahtium*, **37** (June 1983), 3–72.

[203] Brickell, E., "Breaking Iterated Knapsacks" in [48], 342–358.

[204] Chor, Benny; and Ronald Rivest, "A knapsack type public-key cryptosystem based on finite field arithmetic," in [48], 54–65.

[205] Lagarias, J. C., "The computational complexity of simultaneous diophantine approximation problems," *Proceedings of the 23rd FOCS* (November 1982), 32–39.

[206] Lagarias, J. C.; and A. M. Odlyzko, "Solving low-density subset sum problems," *Proceedings of the 24th FOCS* (November 1983), 1–10.

[207] Merkle, Ralph C.; aynd M. E. Hellman, "Hiding information and signatures in trapdoor knapsacks," *IEEE Trans. on Info. Theory*, **IT–24** (September 1978), 525–530.

[208] Odlyzko, A., "Cryptanalytic attacks on the multiplicative knapsack scheme and on Shamir's fast signature scheme," *IEEE Trans. on Information Theory*, **IT-30** (July 1984).

[209] Shamir, Adi, "A polynomial time algorithm for breaking Merkle-Hellman cryptosystems" *Proceedings of the 23rd FOCS* (1982), 145–152.

[210] Shamir, Adi, "On the cryptocomplexity of knapsack systems," *Proceedings of the 11th STOC* (1979), 118–129.

[211] Willett, Michael, "Trapdoor knapsacks without super-increasing structure," Dept. of Mathematics, Univ. of North Carolina at Greensboro (1982).

RSA Cryptosystem and Related Cryptosystems

[212] Alexi, W.; B. Chor; O. Goldreich; and C. P. Schnoor, "RSA/Rabin bits are $1/2 + 1/\text{poly}(\log N)$ secure," *Proceedings of 25th FOCS* (1984), 449–457.

[213] Ben-Or, Michael; Benny Chor; and Adi Shamir, "On the cryptographic security of single RSA bits," *Proceedings of the 15th STOC* (April 1983), 421–430.

[214] Blakley, G. R., "Rivest-Shamir-Adleman public-key cryptosystems do not always conceal messages," unpublished manuscript, Dept. of Mathematics, Texas A&M University (1979).

[215] Blakley, G. R., "Security of number theoretic public-key cryptosystems against random attack I–III," *Cryptologia* (December 1978, January 1979, and April 1979).

[216] Chor, Benny; and Oded Goldreich, "RSA least significant bits are $1/2 + 1/poly(\log n)$ secure," in [48].

[217] Chor, Ben-Zion, *Two Issues in Public-Key Cryptography: RSA Bit Security and a New Knapsack Type Cryptosystem*, MIT Press (1985).

[218] DeLaurentis, John M., "A further weakness in the common modulus protocol for the RSA cryptosystem," *Cryptologia*, **8** (July 1984), 253–259.

[219] DeJonge, Wiebren, "Attacks on some RSA signatures," in [49].

[220] Gardner, Martin, "A new kind of cipher that would take millions of years to break," *Mathematical Games* column, *Scientific American*, **237** (August 1977), 120–124.

[221] Hastad, J., "On using RSA with low exponent in a public-key network" in [49], 403–408.

[222] Herlestam, T., "Critical remarks on some public-key cryptosystems," *BIT*, **18** (1979), 493–496.

[223] Lipton, Richard, "How to cheat at mental poker," Dept. of Computer Science, Univ. of Cal., Berkeley (1979).

[224] Miyaguchi, Shoji, "Fast encryption algorithm for the RSA cryptographic system," *Proceedings of COMPCON* (1982), 632–638.

[225] Rabin, Michael O., "Digitalized signatures and public-key functions as intractable as factorization," technical report TR–212, MIT Lab. for Computer Science, (January 1979).

[226] Rivest, Ronald; Adi Shamir; and Leonard Adleman, "On digital signatures and public-key cryptosystems," *CACM*, **21** (February 1978), 120–126.

[227] Rivest, Ronald L., "A description of a single-chip implementation of the RSA cipher," *Lambda* (fourth quarter, 1980), 14–18.

[228] Rivest, Ronald L., "A short report on the RSA chip," in [46], 327.

[229] Rivest, Ronald L., "Remarks on a proposed cryptanalytic attack on the M.I.T. public-key cryptosystem," *Cryptologia* (January 1978), 62–65.

[230] "Critical remarks on 'Critical remarks on some public-key cryptosystems'," *BIT*, **19** (1979), 274–275.

[231] Rivest, R.; and A. Shamir, "How to expose an eavesdropper," *CACM* (April 1984), 393–395.

[232] Simmons, Gustavus; and Michael J. Norris, "Preliminary comments on the M.I.T. public-key cryptosystem," *Cryptologia* (October 1977), 406–414.

[233] Williams, H. C., "A modification of the RSA public-key cryptosystem procedure," *IEEE Trans. on Info. Theory*, **IT–26** (November 1980), 726–729.

[234] Yuval, G., "How to swindle Rabin," *Cryptologia*, **3** (July 1979), 187–189.

Discrete-Log Cryptosystems

[235] El-Gamal, Taher, "A public-key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Info. Theory*, **31** (1985), 469–472.

[236] Long, Douglas L.; and Avi Widgerson, "How discreet is the discrete log?," *Proceedings of the 15th STOC* (April 1983), 413–420.

Pseudo-Random Bit Generators

[237] Blum, Lenore; Manuel Blum; and Michael Shub, "Comparison of two pseudo-random number generators," in [46], 61–78.

[238] Fairfield, R. C.; R. L. Mortenson; and K. B. Coulthart, "A LSI random number generator (RNG)," *Proceedings of Crypto 84*, to appear.

[239] Kaliski, Burton S., "A pseudo-random bit generator based on elliptic logarithms," presented at Crypto 86 (August 1986).

[240] Lagarias, Jeff; and Jim Reeds, "Extrapolation of nonlinear recurrences," *Proceedings of Crypto 84*, to appear.

[241] Plumstead, Joan B., "Inferring a sequence generated by a linear congruence," *Proceedings of the 23rd FOCS* (November 1982), 153–159.

[242] Shamir, Adi, "On the generation of cryptographically strong pseudo-random sequences," *Proceedings of ICALP* (1981), 544–550.

[243] Vazirani, Umesh V.; and Vijay V. Vazirani, "Efficient and secure pseudo-random number generation," *Proceedings of the 25th FOCS*, 458–463.

Feedback Shift Registers

[244] Gifford, David K.; John M. Lucassen; and Stephen T. Berlin, "The application of digital broadcast communication to large scale information systems," *IEEE Journal on Selected Areas in Communications*, **3** (May 1985), 457–466.

[245] Golomb, Solomon, *Shift Register Sequences*, Aegean Park Press (Laguna Hills, CA, 1982).

Other Cryptosystems

[246] Blum, Manuel; and Shafi Goldwasser, "An efficient probabilistic public-key encryption scheme which hides all partial information," *Proceedings of CRYPTO 84*, to appear.

[247] Goldwasser, Shafi; and Silvio Micali, "A bit by bit secure public-key cryptosystem," technical memo UCB/ERL M81/88, Univ. of Cal., Berkeley (December 1981).

[248] Goldwasser, Shafi; Silvio Micali; and Po Tong, "Why and how to establish a private code on a public network," *Proceedings of the 23rd FOCS* (November 1982), 134–144.

[249] Hill, Lester Sanders, "Cryptography in an algebraic alphabet," *American Mathematical Monthly*, **36** (June-July 1929), 306–312.

[250] Hill, Lester Sanders, "Concerning certain linear transformation apparatus of cryptography," *American Mathematical Monthly*, **38** (March 1931), 135–154

[251] Kruh, Louis, "The genesis of the Jefferson/Bazeries cipher device," *Cryptologia*, **5** (October 1981), 193–208.

[252] McEliece, R.J., "A public-key cryptosystem based on algebraic coding theory," Deep Space Network Progress Report 42–22, Pasadena Jet Propulsion Labs. (January-February 1978), 114–116.

[253] Pless, Vera, "Encryption schemes for computer confidentiality," unpublished manuscript (May 1975).

[254] Reeds, J. A.; and P. J. Weinberg, "File security and the Unix crypt command," *AT&T Bell Labs. Technical Journal*, **63** (October 1984), 1673–1682.

[255] Sherman, Alan T., "On the Enigma cryptograph and formal definitions of cryptographic strength," master's thesis, MIT Department of EECS (June 1981).

[256] Sloane, N. J. A., "Encrypting by random rotations," technical memorandum (1983).

[257] Wagner, Neal R.; and Marianne R. Magyarik, "A public-key cryptosystem based on the word problem," *Proceedings of Crypto 84*, to appear.

[258] Yagisawa, Masahiro, "A new method realizing public-key cryptosystem," unpublished document (1983).

General Cryptanalysis

[259] Friedman, William F., *Elements of Cryptanalysis*, Aegean Park Press (Laguna Hills, CA, 1976).

[260] Friedman, William F., *Military Cryptanalysis*, Agean Park Press (Laguna Hills, CA, 1980).

[261] Hellman, Martin E., "A cryptanalytic time-memory tradeoff," *IEEE Trans. on Information Theory*, **IT-26** (1980), 401–406.

[262] Hitt, Parker, *Manual for the Solution of Military Ciphers*, Aegean Park Press (Laguna Hills, CA, 1976).

[263] Kullbach, Solomon, *Statistical Methods in Cryptanalysis*, Aegean Park Press (Laguna Hills, CA 1976).

[264] Schroeppel, Richard; and Adi Shamir, "A $T \cdot S^2 = O(2^n)$ time/space tradeoff for certain NP-complete problems," *Proceedings of the 20th FOCS* (1979) 328–336.

[265] Sinkov, Abraham, *Elementary Cryptanalysis, A Mathematical Approach*, The Mathematical Association of America (Washington, D.C., 1966).

Digital Signatures

[266] Davies, D. W., "Applying the RSA digital signature to electronic mail," *Computer*, **16** (February 1983), 55–62.

[267] Davies, D.; and W. Price, "The application of digital signatures based on public-key cryptosystems," *Proceedings of the Fifth International Computer Communications Conference* (October 1980), 525–530.

[268] Even, S.; and Y. Yacobi, "Relations among public-key signature systems," technical report TR–175, Computer Science Dept., Technion (1980).

[269] Goldwasser, Shafi; Silvio Micali; and Ronald Rivest,, "A 'paradoxical' solution to the signature problem," *Proceedings of the 25th FOCS* (1984), 441–448.

[270] Goldwasser, Shafi; Silvio Micali; and Andy Yao, "Strong signature schemes," *Proceedings of the 15th FOCS* (1982), 431–439.

[271] Lamport, L., "Constructing digital signatures from a one-way function," technical report CSL-98, SRI International (October 1979).

[272] Ong, H.; C. P. Schnorr; and Adi Shamir, "An efficient signature scheme based on quadratic equations," *Proceedings of the 16th STOC* (1984), 208–216.

[273] Rabin, M., "Digitalized signatures" in [55], 133–153.

[274] Shamir, Adi, "A fast signature scheme," technical memorandum TM–107, MIT Laboratory for Computer Science (1978).

[275] Shamir, Adi., "Identity-based cryptosystems and signature schemes," technical report, Weizmann Institute, Israel (March 1984).

Protocols

[276] Ben-or, M.; O. Goldreich; S. Micali; and R. Rivest, "A fair protocol for signing contracts," *Proceedings of ICALP* (1985), 43–52.

[277] Blum, Manuel, "Coin flipping by telephone," *Proceedings of IEEE COMPOM* (1982), 133–137.

[278] Blum, Manuel, "How to exchange (secret) keys," *ACM Trans. on Computer Systems*, **1** (May 1983), 175–193.

[279] Chaum, David, "Untraceable electronic mail, return addresses, and digital pseudonyms," *CACM*, **24** (1981), 84–88.

[280] Chaum, David, "Security without identification: transaction systems to make big brother obsolete," CACM, **28** (October 1985), 1030–1044.

[281] Cohen, J.; and M. Fischer, "A robust and verifiable cryptographically secure election scheme," *Proceedings of 26th FOCS* (1985), 372–382.

[282] DeMillo, R.; N. Lynch; and M. Merritt, "Cryptographic protocols," *Proceedings of the14th STOC* (1982), 383–400.

[283] DeMillo, Richard; Nancy A. Lynch; and Michael J. Merritt, "Cryptographic protocols," *Proceedings of the 14th STOC* (May 1982), 383–400.

[284] Dolev, Danny; Shimmon Even; and Richard Karp, "On the security of pin-pong protocols," in [46], 177–186.

[285] Dolev, D.; and A. C. Yao, "On the security of public-key protocols," *Proceedings of the 22nd FOCS* (October 1981), 350–357.

[286] Even, S.; and O. Goldreich; "On the security of multi-party ping-pong protocols," *Proceedings of the 24th FOCS* (November 1983), 34–39.

[287] Even, S.; O. Goldreich; and A. Lempel, "A randomized protocol for signing contracts," *CACM*, **28** (June 1985), 637–647.

[288] Feldman, P.; and S. Micali, "Byzantine agreement in constant expected time (and trusting no one)," *Proceedings of the 26th FOCS* (1985), 267–276.

[289] Goldwasser, Shafi; Silvio Micali; and C. Rackoff, "The knowledge complexity of interactive proof-systems," *Proceedings of the 17th STOC* (1985), 291–304.

[290] Luby, M.; S. Micali; and C. Rackoff, "How to simultaneously exchange a secret by flipping a symmetrically biased coin," *Proceedings of the 24th FOCS* (1983), 11–22.

[291] Needham, Roger; and Michael D. Schroeder, "Using encryption for authentication in large networks of computers," *CACM*, **21** (December 1978), 993–999.

[292] Shamir, Adi; Ronald L. Rivest; and Leonard Adleman, "Mental poker," *The Mathematical Gardner*, D. Klarner, ed., Wadsworth (Belmont, CA, 1981), 37–43.

[293] Yao, Andrew C., "Protocols for secure computations," *Proceedings of the 23rd FOCS* (November 1983), 160–164.

Practical Security

Computer Security

[294] Cornwall, Hugo, *Hacker's Handbook*, Century Communications (1985).

[295] Grampp, F. T.; and R. H. Morris, "UNIX operating system security," *AT&T Bell Labs. Technical Journal*, **63** (October 1984), 1649–1671.

[296] Landwehr, Carl, "Formal models for computer security," *ACM Computing Surveys* (September 1981), 247–278.

[297] Saltzer, Jerome H.; and Michael D. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, **63** (September, 1975), 1278–1308.

[298] Morris, R.; and K. Thompson, "Password security: a case history," *CACM*, **22** (November 1979), 594–597.

Physical Security

[299] Chaum, David, "Design concepts for tamper-responding systems," unpublished manuscript (February 1981).

[300] Cunningham, John E., *Security Electronics*, Howard W. Sams (Indianapolis, Indiana, 1983).

[301] Robinson, Robert L., *Complete Course in Professional Locksmithing*, Nelson-Hall (Chicago, 1973).

[302] Sloan, Eugene A., *The Complete Book of Locks, Keys, Burglar and Smoke Alarms*, William Morrow (New York, 1977).

[303] Wire, Eddie The, *The Complete Guide to Lock Picking*, Loompanics Unlimited (Mason, MI, 1981).

Other Topics

Combining Cryptosystems

[304] Asmuth, C. A.; and G. R. Blakley, "An efficient algorithm for constructing a cryptosystem which is harder to break than two other cryptosystems," *Comp. & Maths. with Appls.*, **7** (1981), 447–450.

[305] Even, S.; and O. Goldreich, "On the power of cascade ciphers" *ACM Trans. on Computer Systems*, **3** (May 1985), 108–116.

[306] Luby, Michael; and Charles Rackoff, "Pseudo-random permutation generators and cryptographic composition," *Proceedings of 18th STOC* (May 1986), 356–363.

[307] Merkle, Ralph C., and Martin E. Hellman, "On the security of multiple encryption," *CACM*, **24** (July 1981), 465–467.

Natural Random Bit Generation

[308] Blum, Manuel, "Independent unbiased coin flips from a correlated biased source: a finite state markov chain," *Proceedings of 25th FOCS* (1984), 425–433.

[309] Chor, B.; and O. Goldreich, "Unbiased bits from sources of weak randomness and probabilistic communication complexity," *Proceedings of the 26th FOCS* (1985), 429–442.

[310] Elias, Peter, "The efficient construction of an unbiased random sequence," *Annals of Math. Statistics*, **43** (1972), 865–870.

[311] Maddocks, R. S., *et al.*, "A compact and accurate generator for truly random binary digits," *Journal of Physics E: Scientific Instruments*, **5** (1972), 542–544.

[312] Von Neumann, J., "Various techniques for use in connection with random digits," in *Von Neuman's Collected Works*, Pergamon (1963), 768–770.

[313] Santha, M.; and U. V. Vazirani, "Generating quasi-random sequences from slightly-random sources," *Proceedings of the 25th FOCS* (1984), 434–440.

[314] Vazirani, U. V., "Towards a strong communication complexity theory, or generating quasi-random sequences from slightly-random sources," *Proceedings of the 17th STOC* (1985), 366–378.

Secret Sharing

[315] Chor, B.; S. Goldwasser; S. Micali; and B. Awerbuch; "Verifiable secret sharing and achieving simultaneity in the presence of faults," *Proceedings of 26th FOCS* (1985), 383–395.

[316] Shamir, A., "How to share a secret," *CACM*, **22** (November 1979), 616–613.

Other Works

[317] Alpern, B.; and F. B. Schneider, "Key exchange using 'Keyless Cryptography'," *Information Processing Letters*, **16** (1983), 79–81.

[318] Baldwin, Robert W.; and Alan T. Sherman, "How we solved the $100,000 Decipher Puzzle," presented at the "Rump Session" of the Eurocrypt 85 conference (Linz, Austria, April 1985).

[319] *Data Ciphering Processors Am9518, Am9568, AmZ8068 Technical Manual*, Advanced Micro Device, Inc. (1984).

[320] Gerhart, L.; and R. Dixon, eds., *Special Issue on Spread Spectrum Communications*, *IEEE Trans. on Communications*, COM-25 (August 1977).

[321] *IBM Personal Computer Technical Reference* (July 1982).

[322] Rivest, R. L.; L. Adleman; and M. L. Dertouzos, "On Data Banks and Privacy Homorphisms" in [55], 169–180.

[323] Rivest, Ronald L.; and Alan T. Sherman "Randomized encryption techniques" in [46], 145–163.

[324] Shamir, Adi, "On the power of commutativity in cryptography," unpublished paper, MIT Department of Mathematics (July 1980).

[325] Shannon, Claude E., "Prediction and Entropy of Printed English," *Bell System Tech. Journal* (January 1951), 50–64.

Works on VLSI

Basic Sources

Bibliographies

[326] Cleemput, W. M., "Computer aided design of digital systems: a bibliography," Computer Science Press, vols. I–III (1978).

[327] Rosenberg, Arnold L., "References to the literature on VLSI algorithmics and related theoretical issues" in [337].

See also the bibliography in [345].

Major Conferences

- *Caltech Conference on VLSI.*

[328] Seitz, Charles L., ed., *Proceedings of Caltech Conference on Very Large scale Integration*, Computer Science Dept., California Institute of Technology (January 1979).

[329] Seitz, Charles E., ed., *Proceedings of the Second Caltech Conference on Very Large Scale Integration*, Computer Science Dept., California Institute of Technology (1981).

[330] Bryant, Randal, ed., *Third Caltech Conference on Very Large Scale Integration*, Computer Science Press (1983).

- *Carnegie-Mellon University Conference on VLSI Systems and Computations.*

[331] Kung, H. T.; Bob Sproull; and Guy Steele, eds., *Carnegie-Mellon University Conference on VLSI Systems and Computations*, Computer Science Press (Rockville, MD, October 1981).

- *Chapel Hill Conference on Very Large Scale Integration.*

[332] Fuchs, Henry, ed., *1985 Chapel Hill Conference on Very Large Scale Integration*, Computer Science Press (Rockville, MD 1985).

- *MIT Conference on Advanced Research in VLSI.*

[333] Penfield, Paul Jr., ed., *Proceedings, Conference on Advanced Research in VLSI, January, 1982*, Artech House (Dedham, MA, January 1982).

[334] Penfield, Paul Jr., ed., *Proceedings, Conference on Advanced Research in VLSI 1984*, Artech House (Dedham, MA, 1984).

[335] Leiserson, Charles E., ed., *Advanced Research in VLSI: Proceedings of the Fourth MIT Conference*, MIT Press (1986).

• *Annual Allerton Conference on Communication, Control, and Computing* (1961 to date).

• *ACM IEEE Annual Design Automation Conference (DAC)* (1964 to date).

• *Annual Symposium on Foundations of Computer Science (FOCS)*, sponsored by the IEEE Computer Society's Technical Committee on Mathematical Foundations of Computing (1960 to date).

• *IEEE International Conference on Computer-Aided Design.*

• *IEEE International Conference on Circuits and Computers (ICCC)*, sponsored by the IEEE Computer Society and the IEEE Circuits and Systems Society (1981 to date).

• *International Symposium on Circuits and Systems (ISCS)* (formerly, *International Symposia on Circuits and Systems*), sponsored by the IEEE (1974 to date).

• *Annual ACM Symposium on Theory of Computing (STOC)*, sponsored by the ACM Special Interest Group for Automata and Computability (1969 to date).

Collections and Other Conferences

[336] Duff, I. S.; and G. W. Stewart, *Sparse Matrix Proceedings 1978*, Society for Industrial and Applied Mathematics (1979).

[337] Entenman, George; *et al.*, "Course projects on VLSI algorithmics: 1983," technical report 83–06, Microelectronics Center of North Carolina (July 1983).

[338] Gray, John P., ed., *VLSI 81: Very Large Scale Integration*, Academic Press (London, 1981).

Survey Works

[339] Glasser, Lance; and Daniel W. Dobberpuhl, *The Design and Analysis of VLSI Circuits*, Addison-Wesley (Reading, MA, 1985).

[340] Gray, J. P., "Introduction to silicon compilation," *Proceedings of the 16th DAC* (1979), 305–306.

[341] Mead, Carver; and Lynn Conway, *Introduction to VLSI Systems*, Addison-Wesley (Reading, MA, 1980).

[342] Mukherjee, Amar, *Introduction to nMOS and CMOS*, Prentice-Hall (1986).

[343] Soukup, Jiri, "Circuit layout," *Proceedings of the IEEE*, **69** (October 1981), 1281–1304.

[344] Sze, S. M., *VLSI Technology*, McGraw-Hill (1983).

[345] Ullman, Jeffrey D., *Computational Aspects of VLSI*, Computer Science Press (Rockville, MD, 1984).

[346] Weste, Neil; and Kamran Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*, Addison-Wesley (1985).

Specialized Topics in Mathematics

Graph Theory

[347] Aho, A. V; M. R. Garey; and F. K. Hwang, "Rectilinear Steiner tress: efficient special case algorithms," *Networks*, **7** (1977), 37–58.

[348] Bui, Thang Nguyen, "On bisecting random graphs," MS thesis, Dept. of Computer Science, MIT (January 1983).

[349] Garey, M. R.; and D. S. Johnson, "The rectilinear Steiner tree problem is NP-complete," *SIAM Journal of Applied Mathematics*, **32** (1977), 826–834.

[350] Hopcroft, John; and Robert Tarjan, "Efficient planarity testing," *JACM*, **21** (October 1974), 549–568.

[351] Lipton, Richard J.; and Robert E. Tarjan, "A separator theorem for planar graphs," *SIAM Journal on Applied Math.*, **36** (April 1979), 177–189.

Computational Geometry

[352] Bentley, Jon Louis; and Thomas Ottman, "The complexity of manipulating hierarchically defined sets of rectangles," technical report 81–109, Dept. of Computer Science, CMU (April 1981).

[353] Bentley, Jon Louis; and Derick Wood, "An optimal worst-case algorithm for reporting intersections of rectangles," *IEEE Transactions on Computers*, **C–29** (1980), 571–577.

[354] Guibas, Leo J.; and Jorge Stolfi, "Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams," *Proceedings of the 15th STOC* (April 1983), 221–234.

[355] McCreight, Edward M., "Priority search trees," technical report CSL–81–5, Xerox Palo Alto Research Center (January 1982).

[356] Shamos, M. I.; and D. Hoey, "Closest-point problems," *Proceedings of the 16th FOCS* (1975), 151–162.

Optimization Techniques

[357] Greene, Jonathan W.; and Kenneth J. Supowit, "Simulated annealing without rejected moves," *Proceedings of the 1984 ICCD* (October 1984).

[358] Hajek, Bruce, "A tutorial summary of the theory and applications of simulated annealing," *Proceedings of the 24th Conference on Decision and Control*, (Ft. Lauderdale, FL, December 1985), 757–759.

[359] Hillier, Frederick S.; and Gerald J. Lieberman, *Operations Research*, Holden-Day (San Francisco, 1974).

[360] Johnson, David S.; L. McGeoch; C. Rodriguez; and C. Schevon, "Optimization by simulated annealing: an experimental evaluation," *Workshop on Statistical Physics in Engineering and Biology* (April 1984).

[361] Kirkpatrick, S.; C. D. Gelatt, Jr.; and M. P. Vecchi, "Optimization by simulated annealing," *Science*, **220** (May 1983).

[362] Kirkpatrick, S.; C. D. Gelatt, Jr.; and M. P. Vecchi, "Optimization by simulated annealing," IBM technical report RC 9355 (41093), (April 1982).

[363] Leiserson, Charles E.; and James B. Saxe, "A mixed-integer linear programming problem which is efficiently solvable," *Proceedings of the 23rd Allerton* (October 1983), 204–213.

[364] Lin, Shen, "Some Computer Solutions of the Traveling-Salesman Problem," *Bell System technical Journal*, **44** (1965), 2245–2269.

[365] Romeo, Fabio; and Alberto Sangiovanni-Vincentelli, "Probabilistic Hill Climbing Algorithms: Properties and Applications" in [332], 393–417.

[366] Yao, Frances F., "Efficient dynamic programming using quadrangle inequalities," technical report, Xerox Palo Alto Research Center (1979).

VLSI Theory

VLSI Models

[367] Aggarwal, Alok; Maria Klawe; David Lichtenstein; Nathan Linial; and Avi Wigderson, "Multi-layer grid embeddings," *Proceedings of 26th FOCS* (October 1985), 186–196.

[368] Bilardi, G.; M. Pracchi; and F. P. Preparata, "A critique and an appraisal of VLSI models of computation," in [331], 81–88.

[369] Thompson, C. D., "Area-time complexity for VLSI," *Proceedings of the 11th STOC*, (1979), 81–88.

Lower-Bounds for VLSI Layout

[370] Abelson, H.; and P. Andreae, "Information transfer and area-time tradeoffs for VLSI multiplication," *CACM*, **23** (January 1980), 20–23.

[371] Baudet, G. M., "On the area required by VLSI circuits" in [331], 100–107.

[372] Brown, D.; and Ronald L. Rivest, "New lower bounds for channel width" in [331], 178–185.

[373] Cole, Richard; and Alan Siegel, "On information flow and sorting: new upper and lower bounds for VLSI circuits," *Proceedings of 26th FOCS* (October 1985), 208–221.

[374] Jia-Wei Hong; and H. T. Kung, "I/O complexity: the red-blue pebble game," technical report 81–111, Dept. of Computer Science, CMU (March 1981).

[375] Sahni, S.; and A. Bhatt, "The complexity of design automation problems," *Proceedings of the 17th DAC* (June 1980), 402–411.

[376] Vitányi, Paul M. B., "Area penalty for sublinear signal propagation delay on chip," *Proceedings of 26th FOCS* (October 1985), 197–207.

[377] Vuillemin, Jean, "A combinatorial limit to the computing power of VLSI circuits," *Proceedings of the 21st FOCS* (1980), 294–300.

Systolic Systems

[378] Kung, H. T.; and Charles E. Leiserson, "Systolic arrays (for VLSI)," in [336], 256–282.

[379] Leiserson, Charles, E., *Area-Efficient VLSI Computation*, MIT Press (1983).

Other Works

[380] Bhatt, Sandeep N.; and Frank T. Leighton, "A framework for solving VLSI graph layout problems," technical report TR–305, MIT Lab. for Computer Science (October 1983).

[381] Leighton, Frank T., *Complexity Issues in VLSI: Optimal Layouts for the Shuffle-Exchange Graph and Other Networks*, MIT Press (Cambridge, MA, 1983).

[382] Leiserson, Charles, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Trans. on Computers* (October 1985).

[383] Simonson, Charles, "Graph Embedding Problems," Ph.D. Thesis, Dept. of Computer Science, Northwestern University (June 1986).

Placement

Mincut Placement Techniques

[384] Breuer, M. A., "Min-cut placement," *J. Design Automation and Fault Tolerant Computing*, **1** (October 1977), 343–362.

[385] Ciesielski, Maciej J; and Edwin Kinnen, "Digraph relaxation for CAD layout of cell based integrated circuits," 1983–1984 computer science and computer engineering research review, Univ. of Rochester (1984).

[386] Corrigan, L. I., "A placement capability based on partitioning," *Proceedings of the 16th DAC* (1979), 406–413.

[387] Dunlop, Alfred E.; and Brian W. Kernighan, "A placement procedure for polycell VLSI circuits," *Proceedings of the IEEE Conference on Computer Aided Design* (September 1983), 51–52.

[388] Fiduccia, C. M.; and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," *19th DAC* (1982), 175–181.

[389] Günther, T., "Die räumliche Anordung von Einheiten mit Wechselbezeiehungen," *Elektronische Datenverarbeitung* **6** (1969), 209–212.

[390] Mild, M; and J. O. Piednoir, "Efficient Placement Algorithms for VLSI," *VLSI Design*, (April 1985) 46–50.

[391] Kernighan, B. W.; and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal* (February 1970), 291–307.

[392] Koss, Michael C., "Optimal leaf cell layout using the min-cut heuristic," SB/MA Thesis, Dept. of EECS, MIT (August 1982).

[393] Lauther, Ulrich, "A min-cut placement algorithm for general cell assemblies based on a graph representation," *16th DAC* (June 1979), 1–10.

[394] Yannakakis, Mihalis, "A polynomial algorithm for the mincut linear arrangement of tress," *Proceedings of the 24th FOCS* (1983), 274–281.

Other Placement Techniques

[395] Baker, Brenda; and Ron Y. Pinter, "An algorithm for the optimal placement and routing of a circuit within a ring of pads," *Proceedings of the 24th FOCS* (November 1983), 360–370.

[396] Baker, Brenda S.; E. G. Coffman Jr.; and Ronald L. Rivest, "Orthogonal packing in two dimensions," technical report TRCS79–1, Dept. of Computer Science, Univ. of Cal., Santa Barbara (1979).

[397] Chow, Chee-Seng, "Phonex: An interactive hierarchical topological floorplanning placer," master's thesis, Department of Electrical Engineering and Computer Science, MIT (June 1985).

[398] Dolev, Danny; and A. Siegel, "The separation required for arbitrary wiring barriers," unpublished manuscript, Dept. of Computer Science, Stanford Univ. (April 1981).

[399] Dunlop, Alfred E.; and Brian W. Kernighan, "Automatic layout of gate arrays," *IEEE Symposium on Circuits and Systems*, (May 1983), 1245–1248.

[400] Leiserson, Charles E.; and Ron Y. Pinter, "Optimal placement for river routing," *SIAM Journal on Computing*, **12** (August 1983), 447–462.

[401] Preas, B. T., "Placement and routing algorithms for hierarchical integrated circuit layout," Ph.D. thesis, Stanford Univ. (1979).

[402] Preas, B. T.; and W. M. van Cleemput, "Placement algorithms for arbitrarily shaped blocks," *Proceedings of the 16th DAC* (1979), 474–480.

[403] Preas, B. T.; and C. W. Gwyn, "General hierarchical automatic layout of custom VLSI circuit masks," *J. Design Automation and Fault-Tolerant Computing*, **3** (1979), 41–58.

[404] Schweikert, Daniel, "A 2-dimensional placement algorithm for the layout of electrical circuits," *Proceedings of the 13th DAC* (1976), 408–416.

[405] Siegel, Alan, "The optimal offset problem for river routing," unpublished manuscript, Dept. of Computer Science, Stanford Univ. (February 1983).

[406] Siegel, Alan, "Fast optimal placement for river routing," unpublished manuscript, Dept. of Computer Science, Stanford Univ. (February 1983).

[407] Siegel, Alan; and Danny Dolev, "Some geometry for general river routing," unpublished manuscript, Dept. of Computer Science, Stanford Univ. (February 1983).

[408] Stockmeyer, L. J., "Optimal orientations of cells in slicing floorplan designs," *Information and Control* **57** (1983).

[409] Wong, Geoffrey E., "Analog integrated circuit placement optimization by simulated annealing," master's thesis, Dept. of EECS, MIT (1984).

Routing

Channel Routing

[410] Baker, Brenda S.; Sandeep N. Bhatt; and Frank T. Leighton, "An approximation algorithm for Manhattan routing," *Proceedings of the 15th STOC* (April 1983), 477–486.

[411] Berger, Bonnie Anne, "New Upper Bounds for two-layer channel routing," MA thesis, Dept. of EECS, MIT (January 1986).

[412] Burnstein, Michael, "hierarchical wire routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, (October 1983).

[413] Burnstein, Michael, "hierarchical channel router," *Proceedings of the 20th Design Automation Conference* (1983).

[414] Deutsch, D. N., "Compacted Channel Routing," *Proceedings of the ICCAD*, (November 1985), 223–225.

[415] Frank, András, "Disjoint paths in a rectilinear grid," *Combinatorica*, **2** (November 1982), 361–371.

[416] Hamachi, Gordon T., and John K. Ousterhout, "A switchbox router with obstacle avoidance," *Proceedings of the 21st Design Automation Conference* (1984), 173–179.

[417] Hashimoto, Akihiro; and Jammes Stevens, "Wiring routing by optimizing channel assignment within large apertures," *Proceedings of the 8th DAC* (1971), 155–169.

[418] Leong, H. W.; D. F. Wong; and C. L. Liu, "A simulated-annealing channel router," (November 1985), 226–228.

[419] Luk, W. K., "A greedy switch-box router," technical report: VLSI document V158, Dept. of Computer Science, Carnegie-Mellon University (May 1984).

Other Works on Routing

[420] Brady, Martin L.; and Donna J. Brown, "Arbitrary planar routing with four layers," *Proceedings of the 1984 MIT* (January 1984), 194–201.

[421] Dolev, Danny; Kevin Karplus; Alan Siegel; Alex Strong; and Jeffrey D. Ullman, "Optimal wiring between rectangles," *Proceedings of the 13th STOC* (1981), 312–317.

[422] Eustace, Robert Alam, "Intra region routing," Ph.D. Thesis, Dept. of Computer Science, Univ. of Central Florida (August 1984).

[423] Joobbani, Rostam; Daniel P. Siewiorek; and Sarosh N. Talukdar, "Application of knowledge-based expert systems to detailed routing of VLSI chips," *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers* (October 1985), 199–202.

[424] Kramer, Mark R.; and January van Leeuwen, "Wire routing is NP-complete," technical report RUU–CS–82–4, Dept. of Computer Science, Univ. of Utrecht (February 1982).

[425] LaPaugh, Andrea S., "Algorithms for integrated circuit layout: an analytic approach," Ph.D. thesis, Dept. of EECS, MIT (1980).

[426] Larson, Richard C.; and Victor O. K. Li, "Finding minimum rectilinear distance paths in the presence of barriers," *Networks*, **11** (1981), 285–304.

[427] Lee, C. Y, "An algorithm for path connection and its applications," *IRE Transactions on Electronic Computers*, **EC–10** (September 1961), 346–365.

[428] Leiserson, Charles; and Miller Maley, "Algorithms for routing and testing routability of planar VLSI layouts," *Proceedings of the 17th STOC* (1985), 69–78.

[429] Stenstrom, J. R.; and Robert M. Mattheyses, "Switch-box routing the greedy way," *Proceedings of the ICCAD Conference* (November 1985), 307–311.

[430] Mattison, Roland L., "A high quality, low cost router for MOS/LSI," *Proceedings of the 9th DAC* (1972), 94–103.

[431] Pinter, Ron Y., "The impact of layer assignment methods on layout algorithms for integrated circuits," Ph.D. thesis, Dept. of EECS, MIT, (1982).

[432] Preparata, Franco P.; and Witold Lipski, Jr., "Three layers are enough," *Proceedings of the 23rd FOCS* (November 1982), 350–357.

[433] Rivest, Ronald L.; and C. M. Fiduccia, private correspondence (1983).

[434] Syed, Zahir; Abbas El Gamal; and M. A. Breuer, "On routing for custom integrated circuits," *Proceedings of the 19th DAC* (July 1982), 887–893.

[435] Tompa, Martin, "An optimal solution to a wire-routing problem," *JCSS*, **23** (October 1981), 127–149.

[436] Vecchi, M. P.; and S. Kirkpatrick, "Global wiring by simulated annealing," *IEEE Transactions on Computer-Aided Design*, **2** (October 1983), 215–222.

Compaction

[437] Dunlop, A. E., "Slip: symbolic layout of integrated circuits with compaction," *Computer Aided Design*, **10** (November 1978), 387–391.

[438] Hsueh, Min-Yu; and Donald O. Pederson, "Computer-aided layout of LSI circuit blocks," *Proceedings of the 1979 ISCS* (1979), 474–477.

[439] Kedem, Gershon; and Hiroyuki Wantanabe, "Optimization techniques for IC layout and compaction," technical report 117, Dept. of Computer Science, Univ. of Rochester (September 1982).

[440] Lengauer, T., "Efficient algorithms for the constraint generation for integrated circuit layout compaction," *Proceedings of the 9th Workshop on Graphtheoretic Concepts in Computer Science* (June 1983).

[441] Lengauer, T., "The complexity of compacting hierarchically specified layouts of integrated circuits," *Proceedings of the 23rd FOCS* (November 1982), 358–368.

[442] Maley, Maley, "Compaction with Automatic Jog Introduction," *Chapel Hill Conference on VLSI*, Computer Science Press (Rockville, MD, 1983), 261–283.

The PI System

[443] Baratz, Alan, "A graph theoretic VLSI layout procedure," Ph.D. Thesis, Dept. of EECS, MIT (August, 1981).

[444] Koschella, James J., "A placement/interconnect channel router: cutting your PI into slices," BA thesis, Dept. of Computer Science, MIT (May 1981).

[445] Stenstrom, Ross; and Robert M. Mattheyses, private correspondence (October 8, 1985).

[446] Moulton, Andrew S., "Routing the power and ground wires on a VLSI chip," MA Thesis, Dept. of EECS, MIT (February 1984).

[447] Moulton, Andrew, "Laying the power and ground wires on a VLSI chip," *Proceedings of the 20th DAC* (1983), 754–755.

[448] Novick, Mark, "Algorithms for crossing placement in VLSI design," BA thesis, Dept. of Computer Science, MIT, (June 1985).

[449] Rivest, Ronald L.; Alan E. Baratz; and Gary Miller, "Provably good channel routing algorithms" in [331], 153–159.

[450] Rivest, Ronald L.; and C. M. Fiduccia, "A greedy channel router," *Proceedings of the 19th DAC*, (June 1982), 418–424.

[451] Rivest, Ronald L.; and C. M. Fiduccia, "An algorithm for optimal crossing placement in VLSI design," unpublished manuscript (1983).

[452] Rivest, Ronald L., "The PI (placement and routing) system," *Proceedings of the 19th DAC* (June 1982), 475–481.

[453] Rose, Flavio, "The PI System User's Manual," unpublished document, MIT Lab. for Computer Science (April 1982).

Other Placement and Routing Systems

[454] Hsueh, Min-Yu, "Symbolic layout and compaction of integrated circuits," technical memorandum UCB/ERL/ M79/80 (December 1979).

[455] Johannsen, David L., "Silicon compilation," Ph.D. Thesis, Cal. Institute of Technology, (1981).

[456] Keller, K. H. and A. R. Newton, "KIC2: A low-cost, interactive editor for integrated circuit design," *Proceedings of COMPCON* (1982), 305–306.

[457] Ousterhout, John K., "Caesar: An Interactive Editor for VLSI Layouts," *VLSI Design*, fourth quarter (1981), 34–38.

[458] Ousterhout, John K.; Gordon T. Hamachi; Robert N. Mayo; Walter S. Scott; and George S. Taylor, "A collection of papers on Magic," technical report UCB/CSD 83/154, University of California at Berkeley (December 1983).

[459] Siskin, Jeffrey M.; Jay R. Southand; and Kenneth W. Couch, "Generating custom high performance VLSI designs from succinct algorithmic descriptions" in [333] (1982), 28–39.

[460] Preas, Brian T., and C. S. Chow, "Placement and Routing Algorithms for Topological Integrated Circuit Layout," *Proceedings of the International Symposium on Circuits and Systems* (June 1985).

[461] Sechen, C.; and A. Sangiovanni-Vincentelli, "The Timber Wolf placement and routing package," *Proceedings of the 1984 Custom Integrated Circuit Conference*, (Rochester, May 1984), 522–527.

Layouts of Specific Circuits

[462] Brent, Richard P.; and H. T. Kung, "A regular layout for parallel adders," *IEEE Transactions on Computers*, **C–31** (March 1982), 260–264.

[463] Fischer, M. J.; and M. S. Paterson, "Optimal tree layout," *Proceedings of the 12th STOC* (1980), 177–189.

[464] Guibas, L. J.; H. T. Kung; and C. D. Thompson, "Direct implementation of combinatorial algorithms" in [328], 509–525.

[465] Lingas, Andrzej; Ron Y. Pinter; Ronald L. Rivest; and Adi Shamir, "Minimum edge length decomposition of rectilinear polygons," *Proceedings of the Allerton Conference on Communications, Control, and Computing* **3** (October 1982), 53–63.

[466] Stone, Harold S., "Parallel processing with the perfect shuffle," *IEEE Transactions on Computers*, **C–20** (February 1979), 153–161.

Other Works

[467] Brooks, Frederick P., *The Mythical Man-Month, Essays on Software Engineering*, Addison-Wesley (January 1982).

[468] Moon, David; Richard M. Stallman; and Daniel Weinreb, *Lisp Machine Manual*, 6th edition (June 1984).

[469] Pitman, Kent M., "The revised MacLisp manual," technical report TR–295, MIT Lab. for Computer Science (May 21, 1983).

[470] Zipple, Richard, "Schema," *Proceedings of the DAC* (1985).

About the Author

On February 26, 1957, Alan Theodore Sherman was born in Cambridge, Massachusetts. The son of a college history professor and a high school German teacher, Alan grew up in Williamsburg, Virginia, attending public schools.

In June 1974, Alan graduated as *salutatorian* from Lafayette High School in Williamsburg, Virginia. He then entered Brown University in Providence, Rhode Island, where he pursued a liberal arts course of study. At Brown, he was elected to *Phi Beta Kappa* and *Sigma Xi*. In June 1978, Alan graduated from Brown, *magna cum laude*, with an Sc.B. in mathematics.

Having become interested in theoretical computer science, Alan entered the Ph.D program in computer science in the Department of Electrical and Computer Science at the Massachusetts Institute of Technology in Cambridge, Massachusetts. Working under Professor Ronald Linn Rivest, in June 1981, Alan received a S.M. degree in Electrical Engineering and Computer Science. The title of his master's thesis was "On the Engima cryptograph and formal definitions of cryptographic strength." While at MIT, Alan also completed a graduate minor in computer music. Continuing his research under Professor Rivest, in October 1986, Alan completed all requirements for a Ph.D. in computer science.

Throughout his education, Alan undertook a variety of jobs. From fourth through twelfth grade, Alan played fife in the Colonial Williamsburg Fife and Drum Corps. During summer vacations from college, Alan worked as an interpreter at the Magazine in Colonial Williamsburg and as a programming aide for the Computer Sciences Corporation at the NASA Langley Research Center in Hampton, Virginia. While at MIT, Alan supported himself as a research and teaching assistant; he also served as a private consultant on the application of cryptography. In September 1985, Alan joined the faculty in the Department of Computer Science at Tufts University in Medford, Massachusetts, where he is now an assistant professor.

In spring 1986, Alan won a senior class award for teaching at Tufts University.

Alan's interests outside of computer science include music, board games, and sports. While at MIT, Alan studied piano under Nicholas Van Slyck at the New School of Music and played piano in the MIT Chamber Music Society. During 1980, Alan served as President of MIT's Tang Hall Residents' Association. In his spare time, Alan enjoys chess, tennis, squash, racquet ball, badminton, hiking, skiing, jogging, and ballroom dancing. On August 2, 1986, Alan married Tomoko Shimakawa in Cambridge, Massachusetts.