Automatic Design of Feistel Ciphers Using Constraint Techniques

by
Venkatesh Ramamoorthy

Bachelor of Engineering
in Computer Engineering
University of Bombay, Mumbai, India
1989

Master of Technology
in Computer Science
Indian Statistical Institute, Calcutta, India
1997

A dissertation submitted
to Florida Institute of Technology
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy
in
Computer Science

Melbourne, Florida
July, 2010

© Copyright 2010 Venkatesh Ramamoorthy
All Rights Reserved

The author grants permission to make single copies ____________________

We, the undersigned committee,
hereby approve the attached thesis

Automatic Design of Feistel Ciphers Using Constraint Techniques

by
Venkatesh Ramamoorthy

Marius C.Silaghi, Ph.D.
Dissertation Advisor
Assistant Professor, Computer Science

Philip K.Chan, Ph.D.
Committee Member
Associate Professor, Computer Science

Debasis Mitra, Ph.D.
Committee Member
Associate Professor, Computer Science

Syamal K.Sen, Ph.D.
Committee Member
Professor, Mathematical Sciences

William D. Shoaff, Ph.D.
Associate Professor and Head
Computer Sciences

Abstract

Automatic Design of Feistel Ciphers Using Constraint Techniques

by
Venkatesh Ramamoorthy

Dissertation Advisor: Marius C.Silaghi, Ph.D.

In symmetric key cryptographic algorithms that operate on the Feistel principle, Cryptographic substitution boxes (S-boxes) are employed to introduce confusion into the message being encrypted. These S-boxes constitute the non-linear part in most cryptographic algorithms, and their design has been the focus of attention among researchers for several years. The concerns yield major design requirements. In particular, they should be highly nonlinear. Current work in S-box design to satisfy security requirements employ approaches such as human-made, math-made, generate-and-test, spectral inversion and local search. Recent approaches employ neural networks and distributed methodologies.

This work addresses the application of constraint-based search techniques to find cryptographic substitution boxes (S-boxes). In this approach, variables are defined, the domain of each variable is specified, and common security requirements for an S-box are modeled into constraints involving relevant variables. The model is input to a solver that outputs the S-boxes.

We have made a number of contributions. First, the quality of obtained S-boxes is superior to the ones currently published by the Data Encryption Standard (DES) as part of its specification based on Matsui's security metric. Second, due to the enormity of the problem, several heuristics are investigated for n-ary Constraint Satisfaction Problem (CSP) solvers to speed up S-box search and generation. We apply the properties of CSPs to reduce the search space to obtain solutions both, efficiently and having higher quality according to Matsui's measure for non-linearity. We derive new results on Linear Approximation Tables for an S-box, and on the condition of a partially assigned S-box to form a complete S-box. A method of visiting S-box variables that will efficiently generate S-boxes is identified. A form of value-ordering to propel this efficiency further has been discovered.The properties of constraints are used to discover new forms of symmetry of S-boxes. Finally, a novel metric for search efficiency of systematic searches such as this application has been quantified.

Table of Contents

List of Tables

List of Figures

Dedication

This Dissertation is dedicated to my Father Shri K.Ramamoorthy (April 27,1936 – March 26,2005) without whose encouragement and passion I would never have reached this far in my life.

Acknowledgments

There are several people who I would like to thank during, and before, these Ph.D years. Though I am unable to individually specify each person, I remember Dr. N.Usha Rani who, in my earlier place of work, introduced me to Elementary Cryptography (02-Feb-1998). Given my penchant with numbers, I very soon found myself well into the practical aspects of implementing ciphers, having published three papers a year later in a research setup, thanks to the research initiatives of my managers Dr. Sundeep Oberoi and Dr. Kuttath Ramachandran, and my product managers Mrs.Usha Murthy and Mrs.Lalitha Subramanian at my earlier place of work.

Soon after, my passion for research and teaching was propelled further. As I was getting to that point of figuring out what direction I should consider next, it was my brother Natraj who gave the idea of pursuing my Ph.D degree in the United States. Cryptography was naturally my chosen field of interest.

I joined Florida Tech in August 2004, I made several friends in my campus community. I would like to thank V.L.Praveen who has always been with me, particularly during my starting years when we were room-mates. Whenever I was frustrated, he was always there to bring me food and joy and we'd hang out with our other friends playing cricket. It was a pleasure making it to the kitchen after school! He'd take all the risks, experiment creatively while making food, while I'd assist him in the other peripheral jobs. And we'd always have study sessions during course work and during comprehensive examinations.

One of the courses I took was on Constraint Programming (CSE 5692) offered by Dr. Debasis Mitra in my first-ever semester here (Fall 2004). The only reason I took this course was because I always wanted to take the advanced courses and nothing less. Little did I ever imagine, then, that I was going to be working with this methodology! Dr. Mitra's exercises on the SEND-MORE-MONEY problem for which I had provided two solutions was very interesting from the viewpoint of multiprecision arithmetic that I had worked on prior to landing at Florida!

Dr. Marius Silaghi (my academic advisor) initiated me into this new problem involving applied research. This involved applying Constraint Programming (or Constraint Satisfaction Problem, or CSP) methodologies to searching Cryptographic S-boxes, and got me interested. Dr. Silaghi has been a very gentle guide, always patient. Most importantly, he allows me to understand issues at my own pace by making me work on them, make mistakes (which can be frustrating!), and learn from them. All this, in spite of the time-frame and the difficulties we had to face during the process. He helped me considerably on my first paper. I had the pleasant opportunity to meet his family comprising Dr. Melody Silaghi and the ever-energetic children Melinda, Mircea, Viorel, Ioana and Alina. The children repeatedly challenge me with their mathematical problems and do not let me go away without satisfactory reasoning with them! The Ph.D process has been one of the most educative processes for me, both professionally and for my personal growth, thanks to him and his family.

At around the time I had to undergo an emergency medical operation and would like to especially thank Dr. Saraswathy Ramanathan, her husband Dr. Sridhar Prathikanti, and family. Dr. Saraswathy ensured that I was treated by her group of people, and I was under her care during three weeks of that semester. I would like to thank their parents Dr. K.V.Ramnathan and Mrs.Raji Ramnathan, and Dr. V.Prathikanti and Mrs.Prathikanti for all their help and encouragement during my entire Ph.D process.

Dr. Philip Chan has been my disciplinarian, always ensuring that I kept my focus and never allowing me to lose track. I'm lucky to find compatibility in our ways of working and I thank him for providing me much-needed help. He has been my best trainer from the viewpoint of my teaching philosophy, particularly when I assisted him during the CSE 1001 Java labs. I feel lucky at being able to carry on his encouraging ways of teaching classes.

While my brother Natraj got me started, my father (now late), my mother, and now my spouse Girija (since 2008) and her family, have been supporting me like a rock. My spouse, in particular, has always been resiliently (and smilingly) bearing with my nuances and the behaviors that a typical doctoral student would exhibit particularly during pressure-times. She provides me with the best of foods — both, for my stomach as well as for thought. I thank all of them for their support and encouragement.

Chapter 1
Introduction

Cryptography is the science of hiding information. A reason for this hiding of information is, for example, to ensure its confidentiality. The original message being transformed is called the *plaintext* and the transformed message, the *ciphertext*. The process of transformation is called *encryption* and the reverse process of retrieving the message from its transformed equivalent is called *decryption*. Encryption and decryption of a message is done using *keys*.

The same key that is used to encrypt the message can be used for decryption. This situation is akin to a lock that has one key. Such a cryptographic algorithm is called a *Symmetric* key algorithm. Alternatively, a different (but related) key can be used for decryption from the key used for encryption. Imagine a locking mechanism that can be activated using one key and deactivated using a correspondingly different key. This kind of a cryptographic algorithm is called an *Asymmetric* key algorithm.

Symmetric key algorithms operate on the principle of Feistel ciphers [21]. The Data Encryption Standard (*abbr.* DES) cryptographic algorithm [1] is an example of a Feistel cipher. They operate on the principle of *confusion* and *diffusion*. Diffusion is used to distribute the bits in the input message being encrypted and de-skew the message, and is accomplished using permutations. Confusion is introduced in the message during encryption by substituting parts of the message, replacing them with values. Both are accomplished using *substitution boxes* (S-boxes), which constitutes the most nonlinear transformation in the entire encryption algorithm.

DES is based on eight S-boxes, numbered from S_1 up to S_8, shown in Figure 1.1. Each S-box, organized in the figure as a 4×16 matrix, takes in a six-bit input and gives out a 4-bit output. Thus the total number of input bits to all eight S-boxes taken together is 48, while the total number of output bits is 32.

An example demonstrates DES S-box usage.

Example 1.1 *Consider* 39 *as the 6-bit input to be substituted using DES S-box S_8, i.e., $S_8(39)$ is to be determined. Let $y_0y_1y_2y_3y_4y_5 = 100111_2$ (39 in binary). The row is selected by bits $y_0y_5 = 11_2 = 3$, corresponding to the last row of S-box*

S_1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	3	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	0	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

(a) DES S-boxes S_1 – S_4

S_5

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

(b) DES S-boxes S_5 – S_8

Figure 1.1: Substitution Boxes (S-boxes) S_1 – S_8 used in the Data Encryption Standard (DES)

S_8. The value of $S_8(39)$ is now obtained by indexing into the column $y_1y_2y_3y_4$ of this row, namely, column 0011_2, or column 3. The entry for S-box S_8 in row 3, column 3, read off from Figure 1.1, is 7, that is, $S_8(39) = 7$.

The S-boxes of DES have been the subject of intense speculation by the cryptographic community. Of particular interest has been the manner in which they have been designed, and why the specific numbers shown in Table 1.1 have appeared in the algorithm, particularly due to the classified nature of the design [58]. It was not until after ground-breaking results on differential cryptanalysis by Biham [11], and subsequent results on linear cryptanalysis by Matsui [34], that the design criteria have been published [16]. The requirements for S-box design are specified to ensure maximum security, and a number of them are available in the literature [72, 45, 6, 5, 64, 38].

There are several methods now available to design S-boxes for Feistel ciphers. A classification is made [62] in which S-boxes are generated using random numbers, random generate- and-test, human-made and math-made entries. Recent approaches employ local search, spectral inversion, neural networks and distributed approaches. In most cases, the S-boxes generated need to be checked for satisfaction of the security requirements.

Our work employs a novel approach that uses Constraint Satisfaction Problems (CSPs) to obtain S-boxes. Each security requirement is modeled as a constraint in this approach. The solutions to the CSP are the S-boxes that satisfy the constraints. The main advantage of this approach over the existing ones is that no explicit testing of each S-box for security requirements is necessary since the solutions already satisfy the constraints that model the requirements.

1.1 Problem Statement

S-box Generation Generate a set of cryptographic substitution boxes (S-boxes) that satisfy known design criteria, with each criterion being modeled as a constraint.

Efficiency of Generation Improve upon S-box generation efficiency using this approach.

Quality of S-boxes Generated Provide measures to determine the quality of S-boxes generated using this approach. Improve upon the quality of S-boxes using the properties of CSPs. Compare and contrast with equivalent metrics of S-boxes generated using known approaches.

Arrangements of S-boxes Examine arrangements of S-boxes and examine the overall quality metric of the resulting arrangement.

1.2 Solution Outline

Determining S-boxes can be formulated as a CSP. A CSP consists of variables, domains and constraints. This first step of S-box problem formulation involves identifying the variables, domain of values assumed by each variable, and the constraints connecting the variables.

Next, the model obtained by specifying the variables, domains and constraints is input to a solver. The solver is then executed to yield the S-boxes that satisfy the known constraints. Performance issues are likely to surface and heuristics for speedup are being formulated. An S-box has a particular *size* that should be parameterized. The model should allow for generating lower-size S-boxes, leading to experiments to be performed to examine efficiency of S-box generation using the CSP approach.

To formulate new heuristics, properties of CSPs are applied. In this regard, interesting observations are made. First, a number of constraints are binary, involving two variables, while others are n-ary constraints dealing with more than two variables. Then, some n-ary constraints can be decomposed into lower-arity predicates while others are not straightforwardly decomposable and special forms of projections are being formulated. Aspects of symmetry of constraints are investigated, and their impact assessed on solution speedup.

An important issue is to ascertain the quality of each generated S-box for which a security metric is arrived at, with lower and upper bounds in terms of the S-box size. The special forms of projections for the n-ary constraints includes this security metric.

Finally, an analysis of arranging S-boxes in order to determine an optimum mix is carried out.

1.3 Main Contributions

Our contributions in the CSP-based approach to the design of S-boxes for Feistel Ciphers have been the following.

S-box Generation

1. Using CSPs, we have successfully obtained S-boxes that satisfy the specified criteria, each of which is modeled as a constraint for the CSP.

Efficiency of Generation

1. We have formulated a number of heuristics to improve upon efficiency of S-box generation. The most promising heuristic from the efficiency viewpoint is an incremental, complete heuristic employing a zig-zag variable ordering and permuted domains to generate 6×4 S-boxes.

2. New properties have been formulated and proved for S-box nonlinearity and decomposition of n-ary constraints, to reduce search and speed up S-box generation.

3. New symmetries of S-boxes have been discovered using the CSP methodology, improving efficiency further.

Quality of S-boxes Generated

1. Using an incomplete heuristic, we have obtained 6×4 S-boxes that are superior in quality to those published in [1] depicted in Figure 1.1, as governed by Matsui's metric for the quality of an S-box.

2. A new pattern of visiting S-box entries to speed up the search has been found. A new shuffling of values assigned to S-boxes to not only speed up search but also, improve S-box quality, has been found.

Arrangements of S-boxes

1. An optimum arrangement of a specified number of S-boxes selected from those generated by our model is determined. A metric derived from the differential cryptanalysis of DES [11] is adopted as the certitude of optimality of this arrangement.

1.4 Organization of the Dissertation

The rest of this Dissertation is organized in the following manner. Chapter 2 discusses the S-boxes, where in DES they are located, along with an overview of the evolution of those numbers. It describes relevant research into S-box construction and provides a classification to reveal where our work fits into S-box design. This Chapter also discusses aspects of CSPs relevant to the problem on hand. The CSP solver used in our work in various ways is outlined in an algorithmic fashion at the end of this Chapter.

Chapter 3 discusses our CSP strategy to solve the problem. It presents the modeling of all binary constraints formulated from the S-box requirements.

Chapter 4 models the first of the two n-ary constraints, namely, the nonlinearity constraint. This constraint is straightforwardly implementable as a non-incremental heuristic using a generate-and-test approach, leading to gross inefficiencies. Two incremental heuristics are discussed, one being an incomplete heuristic and the other, a complete heuristic using constraint decomposition. Both incremental heuristics significantly improve upon S-box generation speeds. In addition, the incomplete heuristic has yielded 6×4 S-boxes having nonlinearity metric superior those of the eight DES S-boxes of Figure 1.1.

Chapter 5 models the second of the two n-ary constraints, which we denote in that Chapter as the *COUNT* constraint. This constraint is implementable as a generate-and-test heuristic, which again, leads to inefficient search for S-boxes. An incremental heuristic is presented. The COUNT constraint is not straightforwardly decomposable into constraints of smaller arity. Nevertheless, a projection scheme is employed leading to domain-reduction and adding to efficiency over the incremental heuristic. This is the second, novel heuristic we present in this Chapter.

Chapter 6 discusses aspects of symmetry in S-boxes to contribute to search efficiency, and arrangement of multiple S-boxes to maximize a probabilistic cost function.

Chapter 7 presents experimental results of the various heuristics and their efficiencies. An efficiency analysis is also made varying the size of S-boxes, beginning with smaller-sized S-boxes and on the S-boxes of *Simple DES*, a miniaturized variant of DES provided in [62]. A measure of search progress to quantify efficiency rather than the simple measure of the number of solutions, while generating large-size S-boxes, is formulated. The results of heuristics that improve upon the quality of S-boxes are presented, along with our main contribution of obtaining S-boxes having quality better than the published DES S-boxes as adjudged by Matsui's security metric. We also present orders of efficiencies of variable and value ordering in the experiments.

Chapter 8 discusses the results obtained in Chapter Seven in the light of which heuristic is the most promising, and provides an insight into the nature of the search space. Chapter 9 concludes this Dissertation.

Chapter 2
Background and Related Work

We review the literature related to our topic of research, namely, employing systematic search of cryptographic substitution boxes (*abbr.* S-boxes) using CSPs. The cryptographic algorithm chosen for our experimentation is the Data Encryption Standard (*abbr.* DES) [1]. The reasons for choosing DES are:

1. It is simple to understand and implement, and has in fact been the definitive standard since 1977 for twenty years before the Advanced Encryption Standard (AES) development effort.

2. Extensive research has been done on S-box design of DES-like algorithms and we would like to compare our results against existing work.

3. Eventually an attempt can be made to investigate how the results could be applied — and perhaps, generalized — to the current Advanced Encryption Standard (AES [2]).

In addition, we feel that the mathematical structure of the S-boxes for the AES do not lend themselves for directly investigating S-box search in the AES algorithm. In any case, AES is not a Feistel cipher unlike DES and several other Feistel ciphers, and DES is the right way to begin from.

Our approach to S-box search is a novel one, and to assess against known techniques, one needs to review the following topics, to start with:

1. Existing cryptographic algorithms, where in Cryptography do the S-boxes fit in, and what the properties of these S-boxes are.

2. S-box design. This directly stems from the properties required of an S-box. How S-boxes are currently designed, namely, existing techniques that yield such S-boxes, are studied to analyze how our approach differs from those in the literature to find S-boxes, and also, for more ideas.

3. Existing approaches to S-box search and the their differences from our approach

The relevant literature is classified into the following topics to form the structure for this Chapter.

1. Cryptography
2. S-box design and construction, and
3. Search techniques and CSPs

2.1 Cryptography

The science of Cryptology is classified into two bodies: Cryptography and Cryptanalysis.

2.1.1 Classification

Cryptography is the science of transforming an input data, for transmission or storage, by an entity, into a form that cannot be legible to any entity other than the one transforming the data. This transformation is done with the help of a *key*. The data being transformed is called *plaintext*. The transformed data is called *ciphertext*. The process of transformation is called *encryption* and the key used for transformation is called an *encryption key*. To be able to retrieve the data, the entity applies an inverse transformation and a key. The process of inverse transformation is called *decryption* and the key used for this transformation is called a *decryption key*. The transforming algorithm is called a *cipher*, or Cryptographic Algorithm. Both these terms are used interchangeably in this Dissertation.

Cryptanalysis on the other hand, is the science of deducing the encryption key given parts of the ciphertext, and optionally, the corresponding plaintext.

Figure 2.1 graphically illustrates the different types of cryptographic algorithms. There are two types of ciphers, namely, Stream Ciphers and Block Ciphers. Depending on whether the same key used for encryption is also used for decryption, or otherwise, we also have Asymmetric (or Public) Key Algorithms and Symmetric (or Secret) Key Algorithms.

2.1.2 Stream Ciphers

Stream ciphers are used to encrypt variable-sized data (typically at the bit level) with the help of a variable encryption key. The simplest stream cipher is the exclusive-OR operator, which takes in a stream of plaintext bits and performs a bit-wise exclusive-OR on these bits with a key-stream generated by a key-stream generator, to yield the ciphertext. The same key-stream, when exclusively-ORed

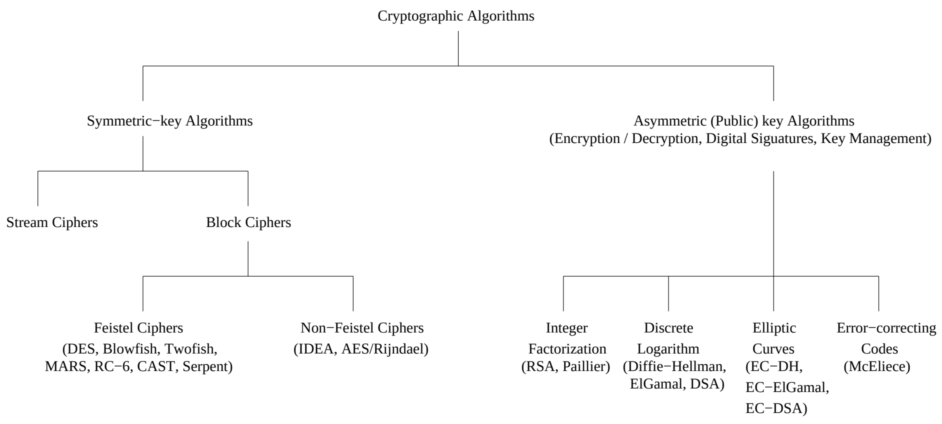

Cryptographic Algorithms
Symmetric−key Algorithms
Asymmetric (Public) key Algorithms (Encryption / Decryption, Digital Siguatures, Key Management)
Stream Ciphers
Block Ciphers
Feistel Ciphers (DES, Blowfish, Twofish, MARS, RC−6, CAST, Serpent)
Non−Feistel Ciphers (IDEA, AES/Rijndael)
Integer Factorization (RSA, Paillier)
Discrete Logarithm (Diffie−Hellman, ElGamal, DSA)
Elliptic Curves (EC−DH, EC−ElGamal, EC−DSA)
Error−correcting Codes (McEliece)

Figure 2.1: A Classification of Cryptographic Algorithms

with the ciphertext will (obviously) yield the plaintext. A more involved stream cipher employs shift registers. They are easily implemented in hardware, are very reliable, can perform at high speeds, and are typically used in military applications. A disadvantage is that the size of the key-stream should be equal to that of the data-stream, which is not practical.

2.1.3 Block Ciphers

Block ciphers encrypt fixed-length blocks of data with the encryption key. For example, a 1-MB file is divided into a number of fixed-length blocks. Each block is encrypted using the key. The block-length and key-length, usually measured in bits, depends upon the cipher. For DES, they both are equal to 64 bits. For the AES, the block-length is 128 bits but Rijndael [18] (of which the AES is a part) additionally supports block-lengths of 160, 192, 224 and 256 bits. The key-length is either of 128, 192 and 256 bits for both, Rijndael and the AES. Block ciphers are used in commercial applications such as banking transactions, disk protection and sector-level encryption. However they suffer from the drawback of cryptanalysis. This happens because the key size is smaller than all of the data being encrypted. For DES, the key is 64 bits long but the data being encrypted can be arbitrarily long, and the same key is repeatedly used for each block the data is divided into. Cryptanalysis exploits this repetitive nature of operations.

2.1.4 Asymmetric (Public) Key Cryptographic Algorithms

In asymmetric key encryption, the key used for decryption is different from that used for encryption. However, they are always related. Asymmetric key cryptosystems are usually put to use in the following way. One of the keys — the one used for encryption — is usually published by the user so that anyone in the world can use to encrypt data to send to this user. The second key is kept private and in the user's custody, used to decrypt the received ciphertext. The encryption key is called the *public key* and the decryption key, the *private key*.

Asymmetric key algorithms are designed in such a way that the public key cannot be used to determine the private key. It is the difficulty of determining this relationship between the keys that protects the cryptanalysis of these algorithms. Rivest-Shamir-Adleman (RSA) [36], ElGamal [58], Paillier [44], McEliece [65] and Elliptic Curve Cryptography (ECC) [62] are some examples of asymmetric key algorithms.

Asymmetric key algorithms (with the exception of McEliece and ECC) are based on number theory and deal with large prime numbers, typically around 4096 bits at most (around 1230 decimal digits). (ECC has around 183-bit key-lengths). The RSA algorithm rests on the Integer Factorization problem while the ElGamal, Diffie-Hellman and DSA [3] algorithms rely on the Discrete Logarithm problem. The Paillier encryption algorithm [44] relies on the Composite Residuosity Class problem. The McEliece encryption algorithm is based on the theory of error-correcting codes [65], [35]. These problems determine the relationship between the public and private keys.

Asymmetric-key algorithms rely on heavy mathematical computation (in particular, modular exponentiation). Algorithms for performing arithmetic on very large numbers, also called *multiprecision* numbers, are discussed in [36]. As such, asymmetric key algorithms are not suitable for encryption of bulk data such as large files. They are used in key management (section 2.1.6).

Asymmetric key algorithms are also used for digital signatures. Signing a document is an operation performed by a user with the private key. The signed document, when sent to a recipient, is first verified by that recipient using the sender's corresponding public key available with the recipient. Since the signer used the private key, which is kept secret with the signer, only the signer and no one else would have signed the document. The RSA encryption algorithm is used for digital signatures as well. For this algorithm, the public and private keys are interchangeable. This is not true in general, for ElGamal has a signature algorithm different from that used for encryption / decryption. Another signature algorithm used is the Digital Signature Algorithm (DSA).

Asymmetric key algorithms assume that the sender already has the recipient's public key. How does the sender first get this public key from a recipient? This topic is mentioned under the topic of key management (section 2.1.6).

ECC relies on the difficulty of finding the abscissae of a point on a curve given its ordinate, modulo a prime. ECC has analogs for the Diffie-Hellman and DSA algorithms, called EC-DH and EC-DSA [28] algorithms, respectively.

2.1.5 Symmetric (or Secret) Key Cryptographic Algorithms

In symmetric key encryption, the same key is used for encryption and decryption. Usually, symmetric key encryption algorithms rely on logical operations such as bit left-shift, right-shift, left and right rotations, exclusive-OR, AND, OR, and NOT operations. They lend themselves naturally towards hardware implementations, while software implementations need to be optimized for high-speed operations. Due to their high-speed operation, symmetric key algorithms are used for bulk data encryption and decryption. DES [1], AES [2], IDEA [32], CAST [4], Blowfish [57], Twofish [59] and several others are examples of symmetric key algorithms.

An assumption inherent in symmetric key encryption and decryption is that the encryption key is already shared between the sender and recipient. How this sharing is done is the subject of key management, discussed in Section 2.1.6.

2.1.6 An Overview of Key Management, Secret Sharing and Security Protocols

Key management is used to solve the assumptions of symmetric and public key cryptosystems.

1. In symmetric key cryptosystems, the two parties involved in secure communications already have the encryption key in place. This is ensured using key negotiation or key agreement.

2. In public key cryptosystems, the sender is assumed to possess the recipient's public key. This is ensured using digital certificates [58].

Key agreement is done using an asymmetric key encryption algorithm. The sender wishing to transmit the symmetric key securely to the recipient encrypts the symmetric key with the recipient's public key and sends the ciphertext to the recipient. Since this symmetric key is only around 64-256 bits long, speed is not an issue. The recipient decrypts the received ciphertext using the private key. Now both parties have the symmetric key available with them, for bulk data encryption. Notice that a protocol has evolved during this process.

Key negotiation is done between the sender and recipient using information common to everyone, and information known only to the sender and only to the recipient (not both). The Diffie-Hellman key negotiation protocol is an example.

With keys being shared between parties involved in communication, arises the idea of how secrets can be shared between the parties. Shamir's secret sharing scheme [52] can be used to split a secret into shares, one for each participant, such that a minimum number of participants (called a *threshold*) only can reconstruct the secret. A different, and somewhat less efficient threshold scheme was developed by Blakley [12]. *Verifiable* secret sharing is proposed in [61] to achieve security against cheating participants. If everyone in the group of participants can verify that the shares are correctly distributed, the scheme is called a publicly verifiable secret sharing scheme.

Operations can also be done on secrets by the group of participants using their shares alone, without knowing what the underlying secrets are. Such operations can include resizing (reducing) the threshold of a share, performing addition, subtraction, scalar multiplication and multiplication of secrets. More operations include generating a random number or a random bit without each user knowing its value, computing the square root of a secret using its shares, finding the multiplicative inverse of a secret, unbounded fan-in (multiplying secrets), and logical operations (AND, OR, NOT, exclusive-OR) [19].

An example of a security protocol is by Needham and Schroeder [42], that has been modeled as a soft CSP over the framework of semirings [8] for confidentiality analysis.

Further discussions on Asymmetric key algorithms, key management and security protocols [62, 58, 36] are outside the scope of this Dissertation.

2.2 Feistel Networks

This section discusses Feistel Networks, their workings (especially DES), and where the S-boxes fit into a symmetric key cipher are now discussed.

2.2.1 Confusion and Diffusion

Symmetric key block ciphers operate on the principles of product ciphers, using confusion and diffusion [62, 58, 40]. Confusion is introduced during transformation to make the relationship between the key and ciphertext as complex as possible. This is usually achieved by substituting parts of plaintext bits with constant bits using substitution boxes (S-boxes).

Diffusion on the other hand is introduced during transformation in order to spread the influence of plaintext characters over as much of the ciphertext as possible. This ensures that statistical properties of the plaintext are hidden in the ciphertext. Diffusion is accomplished by permuting the plaintext bits.

A product cipher composes the operations of confusion and diffusion. Doing so only once is not sufficient, and repeating these operations achieves the desired effect.

2.2.2 The Feistel network as a Product Cipher

The Feistel network, first designed by Horst Feistel, is a product cipher that repeatedly performs the following steps on a block of plaintext: Permute the input bits (diffusion), and apply S-boxes (confusion) on these permuted bits. These steps, called an encryption *round*, also consist of mixing a transformation of the encryption key, called *subkey*. Each round uses a subkey different from the others, and the subkeys are generated from the encryption key using a *key schedule*.

Let a block of plaintext being encrypted be represented by $m = L_0R_0$, where the block m is of length n bits, and L_0 and R_0 are two *halves* of this block, each having length $n/2$ bits. Let also, the key schedule take the encryption key K as input and generate subkeys $K_1, K_2, \ldots K_r$, for the r rounds. A round of encryption is a function f, that takes in two inputs, a subkey K_i and a half-block R_{i-1}, and gives out an $n/2$-bit half-block R_i, $1 \leq i \leq r$. Define half-blocks $L_1, R_1, L_2, R_2, \ldots L_r, R_r$, where for $i = 1, 2, 3, \ldots, r$,

$$L_i = R_{i-1},\ R_i = L_{i-1} \oplus f(K_i, R_{i-1}).$$

Visually, this means that there are r transformations on a half-block, and in each round, a transformation is followed by a swap of the two encrypted half-blocks. After the last round, the swap step is undone and the ciphertext, therefore, is R_rL_r. The situation is depicted in figure 2.2.

An interesting features of the Feistel network is that the decryption algorithm is the same as the encryption algorithm, except that the subkeys are consumed in each round in an order that is the reverse of the order used in encryption. This is good since one does not have to implement separate algorithms for encryption and decryption.

2.2.3 The Data Encryption Standard (DES)

DES is a 16-round Feistel cipher that takes a 64-bit input block of plaintext, a 64-bit key, and outputs a 64-bit output block of plaintext. The key schedule generates 16 48-bit subkeys, each to be used in one round of encryption. An initial permutation (IP) is applied to the 64-bit plaintext block before the 16 rounds begin. For the rounds, each block is divided into a left and right 32-bit half-block that forms L_0R_0 of Figure 2.2.

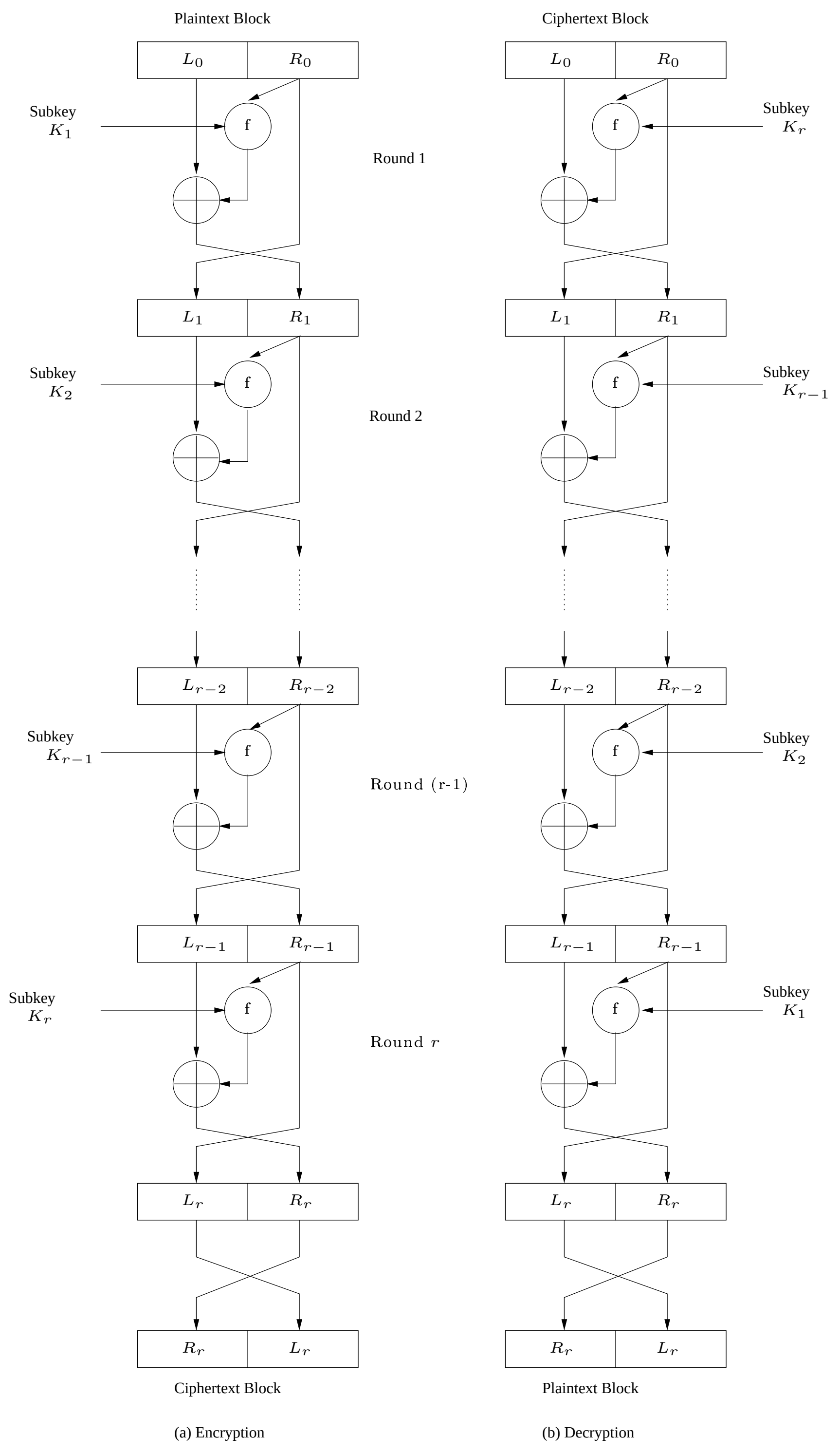


Figure 2.2: Rounds of encryption / decryption in a typical Feistel Cipher

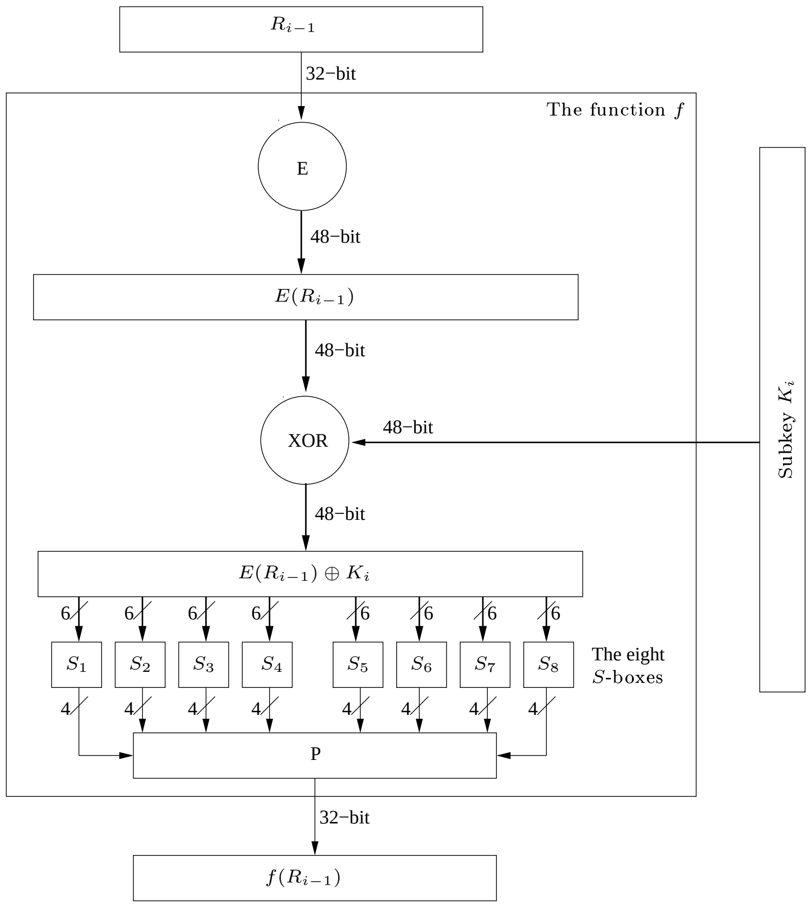


Figure 2.3: The function f of DES that uses S-boxes

2.2.4 The Round Function and the S-boxes of DES

The round function f for DES is shown in Figure 2.3. f consists of an expansion function E, an exclusive-OR operation with the subkey for the current round, an application of eight Substitution Boxes (S-boxes), and a permutation P on the bits.

E stretches a 32-bit half-block by repeating 16 bits once, to yield a 48-bit value. This is exclusively-ORed with the 48-bit subkey in the current round. The 48-bit exclusive-OR result is input to eight S-boxes, each taking in six bits. Each S-box produces a 4-bit output, and the eight 4-bit outputs together form a 32-bit output. This result is permuted using P to yield a 32-bit value, which now exclusively-ORed with the other 32-bit half to give the output half-block of a round of DES encryption. The other half-block (directly from the input) and this output half-block are interchanged before being passed to the next round.

After the last round, no interchange is done, and the inverse permutation IP^{-1} is applied to both half-blocks taken together to yield a 64-bit output ciphertext block.

As shown in Figure 2.3, the round function f contains the S-boxes that are the subject of this Dissertation. The S-boxes of DES that have been published in [1] are shown in Table 1.1 and they consist of fixed numbers. An interesting feature of the Feistel network is that these S-boxes can be replaced by other S-boxes, and the encryption and decryption will still work. Obviously the encryption will produce a different ciphertext now.

2.3 The Design of Substitution Boxes

Each S-box in DES yields a 4-bit output string for a 6-bit input string, suggesting a many-to-one function. The numbers in the DES S-boxes are fixed and are shown in Table 1.1. The S-boxes of DES have been the subject of intense speculation right from their inception. The design principles are the results of years of research by the cryptographic community, particularly after allegations that the NSA may have modified them to introduce a trap-door for the government to intercept messages [58]. The design criteria was classified and were revealed [16] only after results of differential cryptanalysis were published by Eli Biham, and linear cryptanalysis by Matsui [34]. A lucid tutorial of differential and linear cryptanalysis is provided in [26]. Susan Landau [33] discusses these, and in addition, provides a third attack by Wiener. According to the paper, based on exhaustive search but with technology at that time (1993), Wiener estimated that the $1-Million machine with 57,000 DES chips and a pipelined architecture (constantly feeding data and computing simultaneously) could break a DES-encrypted message in 3.5 hours.

2.3.1 The Security Criteria for DES S-boxes

The design requirements of DES, eight in number, are listed in Table 2.1 [16]. These design criteria are labelled **S-1**, **S-2**, and so on upto **S-8**. In our work, we formulate constraints to model the requirements, and input the constraints to a solver. Substitution Boxes are generated by the solver to satisfy the constraints that model the stated requirements. This approach to S-box generation turns out to be a novel one, and no literature has been available so far that relates to this specific approach. Therefore, existing approaches of designing and generating substitution boxes are discussed, and mention is made of how the ideas are being (or will be) applied to our work.

S-1 "Each S-box has six bits of input and four bits of output."

S-2 "No output bit of an S-box should be too close to a linear function of the input bits. (That is, if we select any output bit position and any subset of the six input bit positions, the fraction of inputs for which this output bit equals the exclusive-OR of these input bits should not be close to 0 or 1, but rather should be near $\frac{1}{2}$)."

S-3 "If we fix the leftmost and rightmost input bits of the S-box and vary the four middle bits, each possible 4-bit output is attained exactly once as the middle four input bits range over their 16 possibilities."

S-4 "If two inputs to an S-box differ in exactly one bit, the outputs must differ in at least two bits."

S-5 "If two inputs to an S-box differ in the two middle bits exactly, the outputs must differ in at least two bits."

S-6 "If two inputs to an S-box differ in their first two bits and are identical in their last two bits, the two outputs must not be the same."

S-7 "For any nonzero 6-bit difference between inputs $\Delta I_{i,j}$, no more than eight of the 32 pairs of inputs exhibiting $\Delta I_{i,j}$ may result in the same output difference $\Delta O_{i,j}$."

S-8 "Similar to **S-7**, but with stronger restrictions in the case $\Delta O_{i,j} = 0$ for the case of three active S-boxes on round i."

A criterion, stronger than **S-2**, named as **S-2'** [16] is often quoted:

S-2' "No linear combination of output bits of an S-box should be too close to a linear function of the input bits. (That is, if we select any subset of the four output bit positions and any subset of the six input bit positions, the fraction of inputs for which the exclusive-OR of the output bits equals the exclusive-OR of these input bits should not be close to 0 or 1, but rather should be near $\frac{1}{2}$)."

Table 2.1: The S-box criteria used by IBM for designing DES [16]

2.3.2 S-boxes and Boolean Functions

An $n \times m$ S-box is defined to be a Boolean Mapping $S : \mathbb{Z}_{2^n} \rightarrow \mathbb{Z}_{2^m}$ that takes an n-bit string and gives out an m-bit string. Here $\mathbb{Z}_k$ stands for the set $\{0, ...k-1\}$. S need not be invertible. They are used in the generation of a parameterized substitution of x, through the function

$$f : \mathbb{Z}_{2^m} \times \mathbb{Z}_{2^n} \rightarrow \mathbb{Z}_{2^m}, f(x, y) = x \oplus S(y),$$

where y is the parameter. The obtained substitution function is a parameterized bijection, and it is therefore often referred to as a permutation function.

If $n = m$, S is more likely to be a one-one and onto itself, that is, a permutation of the set $\{x : 0 \leq x < 2^n\}$. This is, in fact, recommended for an $n \times n$ S-box [73].

A Boolean function is defined to be a Boolean Mapping $F : \mathbb{Z}_{2^n} \rightarrow \mathbb{Z}_2$ that takes as input an n-bit string and yields as output either 0 or 1. Thus a Boolean function is an $n \times 1$ S-box and is many-to-one.

2.3.3 S-box Representation

Current literature suggests three ways [38] in which an $n \times m$ S-box $S(x)$, $0 \leq x < 2^n$, is represented. We add a fourth representation that is used in our work.

Function Representation using m-bit numbers In this representation, the mapping is defined as

$$S(x) = r_x, 0 \leq r_x < 2^m. \tag{2.1}$$

Bitwise Representation with Bits from Boolean functions In this representation, each S-box entry is expressed as an aggregation of bits, each in turn generated by a Boolean function:

$$\begin{bmatrix} C_{m-1}(x) & C_{m-2}(x) & \dots & C_0(x) \end{bmatrix}, \tag{2.2}$$

where $0 \leq i < m$ and each $C_i(x) : \mathbb{Z}_{2^n} \rightarrow \mathbb{Z}_2$ is a Boolean function. $C_i(x)$ is regarded in [38] as a “column of the S-box (entry)".

The (decimal) expression formed from the bitwise representation of an S-box, given by

$$S(x) = 2^{m-1}C_{m-1}(x) + 2^{m-2}C_{m-2}(x) + 2^{m-3}C_{m-3}(x) + \ldots + C_0(x),$$

provides for conversion from the bit-wise representation into the functional representation using m-bit numbers given by Equation 2.1, and conversely.

Boolean Matrix An S-box can be represented by a $2^n \times m$ binary matrix

$$[c_{ij}]_{2^n \times m}, \text{ where} c_{ij} \in \mathbb{Z}_2, 0 \leq i < 2^n, 0 \leq j < m. \tag{2.3}$$

$S(x)$ is now obtained as follows:

$$S(x) = \sum_{j=0}^{m-1} c_{xj} 2^{m-j-1}$$

This representation can be viewed as a binary *vector* or *one-dimensional* form of representation of all 2^n entries of an S-box. We now formulate a *two-dimensional* representation that is more useful in our work, in which each entry can be either binary, decimal or hexadecimal.

A Tabular form of S-box Representation In our work, each $n \times m$ S-box is organized as a $2^{n-m} \times 2^m$ table, to contain a total of 2^n entries. The $(n - m)$ bits taken together form the row-select, and the remaining m bits index into the column of the selected row. Each row is a permutation of the m-bit numbers in $\mathbb{Z}_{2^m}$. Thus each S-box has 2^{n-m} such occurrences of these numbers, one in each row. In other words, there exists four inputs that map to the same S-box entry in the table (many-to-one function).

For example, each S-box of DES is typically organized as a table with 4 rows, 16 columns as Figure 1.1 illustrates. The input to each S-box ranges from 0 to 64, while the corresponding output ranges from 0 to 16. Example 1.1 discusses the usage of a DES S-box.

2.3.4 Specification of S-boxes in Cryptographic Algorithms

The DES S-boxes of Figure 1.1 are fixed and specified as part of the algorithm specification [1]. For other algorithms, the S-boxes may be similarly fixed or may be variable, or often, computed depending upon a parameter. Different ways of specifying S-boxes in a cipher are now discussed.

Based upon the different ways in which the S-boxes are specified, in cryptographic algorithms they can be classified as follows:

Fixed S-boxes

These substitution boxes are constant, and are specified as part of the algorithm to be used for encryption / decryption. Several Feistel ciphers specify S-boxes that fall under this category. Examples are DES and CAST-256 [4].

Variable S-boxes

In these algorithms, the entries in the substitution boxes are generated as a part of the encryption / decryption process, and used at that point. They get modified between two encryption rounds. Blowfish [57] is an example where the S-boxes depend on the encryption key. This is done to counter linear and differential cryptanalysis. Another example is Twofish [59]. IDEA [32], although not a Feistel cipher, implements a multiplication step modulo $(2^{16} + 1)$ which is also viewed as an S-box [58], and this operation depends on parts of the key (i.e. a key-dependent S-box). An advantage is a saving on memory tables that would otherwise have to be initialized had the entries been fixed. A disadvantage of this scheme is that for every encryption / decryption session, the S-boxes have to be set up and startup times can become expensive if the operation has to be carried out repeatedly using different keys.

2.3.5 S-box Generation Techniques

Four techniques are outlined for the generation of S-boxes [62, 40].

Random

Generate S-box entries using some pseudo-random number generator, or from a table of random digits. For small-size S-boxes, like the 6×4 S-boxes of DES, this strategy may result in undesirable characteristics, but for those having large size (for example, 64×32 S-boxes, this approach should be acceptable.

A variation of this technique is to initialize S-boxes with pseudo-random digits and as rounds progress, keep changing them depending upon the data and / or the key. This is exactly what is done in Blowfish [57] and Twofish [59].

Random with testing

Randomly select S-box entries, then test the results against various criteria. We find that the level of testing subdivides this technique further.

Generate-and-test every entry

Every S-box entry is randomly generated and tested. In addition, each entry is tested against its neighbors that are the several other entries, within the same S-box. Among the eight design criteria **S-1** to **S-8** of DES [16] (Table 2.1), criteria **S-3** to **S-7** examine neighboring entries in an S-box. This scheme is discussed in detail in Section 2.5.

An obvious extension of testing every entry in a *single* S-box is to test entries across *multiple* S-boxes. This requirement stems from the fact that not all S-boxes can be identical. Testing and rejecting S-boxes by examining their influence on neighboring S-boxes has been recommended [17]. DES constraint **S-8** [16] suggests this requirement for any three out of the eight S-boxes. These tests are apparently formulated to thwart attacks due to differential cryptanalysis [11].

Human-made

This strategy more or less employs a manual approach and the underlying mathematics used to support the same is very simple. The S-boxes of DES were apparently formulated using this approach. For large-size S-boxes the approach becomes impractical. Even for small-sized ones, testing, particularly at the bit-level, can become cumbersome and prone to human errors, with more efforts required to review and correct.

The first S-boxes for Feistel ciphers were designed by hand. Early security attacks have propelled the research for guidelines (i.e., requirements) that avoid known vulnerabilities. These requirements prove to be so difficult to achieve, to the point where it is said [1] that the DES designing team dropped guards when hand-picking their last S-box (given the fact that their last S-box is susceptible to attacks from differential cryptanalysis [11]).

Math-made

Generate S-boxes based on mathematical principles. Using mathematical construction, the resulting S-boxes can be constructed to offer proven security against linear and differential cryptanalysis, together with good diffusion. For example, *Bent* functions 2.4.5 are loaded into S-box entries as part of S-box construction [53, 23, 7, 38].

2.4 Recommended Properties of an S-box

The S-boxes of DES form the only non-linear part of the algorithm [72]. As such, any lacuna in S-box design will severely affect the cipher. After the results of Biham's differential cryptanalysis [11] and Mastui's linear cryptanalysis [34] were published, cryptographers worked on deriving more criteria for S-boxes in order to thwart against these and other attacks.

Several design criteria have been evolved and these are discussed below. In our work, we have not used any of these since our objective is to generate S-boxes that satisfy DES criteria *alone* to begin with. Once that is done, we can improve on search space by modeling each of these requirements into a constraint, adding to the existing set of constraints and narrowing the search space.

2.4.1 Nonlinearity

An obvious requirement is that the S-boxes be nonlinear. This means that no output should be close to a linear combination of any subset of the input bits. The DES design criterion **S-2** in Table 2.1 recommends that the fraction of the number of output bits that is a linear combination of a subset of input bits should be neither 0 nor 1, but close to $\frac{1}{2}$ [16]. Table 2.1 additionally specifies a more stringent, optional criterion labelled **S-2'**. This requirement states that the fraction of a linear combination of a *subset* of output bits to that of the input bits should be close to $\frac{1}{2}$.

Nonlinearity is defined in terms of the distance from the set of all affine functions [45, 23]. It is also defined in terms of the spectrum of a Boolean function (also called the *Walsh Transform*) [23]. Algorithms to construct non-linear Boolean functions and S-boxes using the bit-by-bit approach is provided in [45].

Gupta and Sarkar [23] has modified an elegant algorithm by Zhang and Zheng to generate S-boxes having an extended degree. They have also modified the Maiorana-McFarland technique to generate S-boxes having non-linearity better than previously known construction methods. The S-boxes generated by them are useful for stream ciphers.

2.4.2 Diffusion characteristics of S-boxes

A requirement of a good S-box is the possession of strong diffusion characteristics. This means that changing a small number of its input bits should result in a change in a very large number of its output bits.

2.4.3 Avalanche Criterion, and Strict Avalanche Criterion

A requirement of a good cipher is that in general, complementing one bit of the input should result in a change in an average of half the number of output bits. This requirement is called the *Avalanche Criterion.* A more stringent requirement called the *Strict Avalanche Criterion* (SAC) [72] states that each output bit should change with a probability of $\frac{1}{2}$ whenever a single input bit is complemented. SAC is quantified in [38] in terms of another measure, the *dynamic distance* of an S-box. This paper also defines a *Maximum Order SAC* (MOSAC), and recommends that an ideal S-box should satisfy MOSAC.

There is also a *Guaranteed Avalanche Criterion* (GAC) [64]. An S-box satisfies a GAC of order γ if any single-bit inversion at its input results in at least γ bits of inversion at its corresponding output.

Heys and Tavares [64] discusses the effect of the number of rounds, S-box size, diffusion characteristics on avalanche characteristics. Specifically, the avalanche criterion is satisfied in fewer rounds when the guaranteed avalanche parameter γ increases. When the size of the S-box increases, the avalanche behavior of the encryption network improves.

Seberry, Zhang and Zheng [60] propose a novel systematic scheme of generating S-boxes based on group Hadamard matrices. In particular, their generated S-boxes satisfies the SAC. They have defined a measure of robustness against differential cryptanalysis, and ensured that the S-boxes generated by them satisfy this measure.

2.4.4 Bit Independence Criterion

The Bit Independence Criterion (BIC) [72] states that when any single input bit i is inverted, for all i, j, k, output bits j and k should change independently. The BIC is quantified in [38] in terms of the *distance to higher order BIC* (DHOBIC).

There is another measure quantified in [38], called the *Maximum Order BIC* (MOBIC), defined in terms of the dynamic distance. The paper recommends that an ideal S-box should satisfy MOBIC.

2.4.5 Bent functions

Mister and Adams [38] propose that all linear combinations of S-box columns should be *bent* functions. Bent functions are a special class of Boolean functions that are highly non-linear [7]. They are defined in terms of the spectrum of a Boolean Function (the Walsh Transform) [38], [23]. These are used for S-box construction by the bit-by-bit construction method discussed in subsection 2.3.5.

2.5 Search techniques

While there are a number of search techniques in the discipline of Artificial Intelligence (AI) [54], we can divide the approaches to S-box search, using AI, into two simple classes for our purposes, namely, Nonsystematic Search and Systematic Search. In this section, we focus on S-boxes generated in the available literature, and classify them into the appropriate AI category. Figure 2.4 summarizes the taxonomy of S-box search.

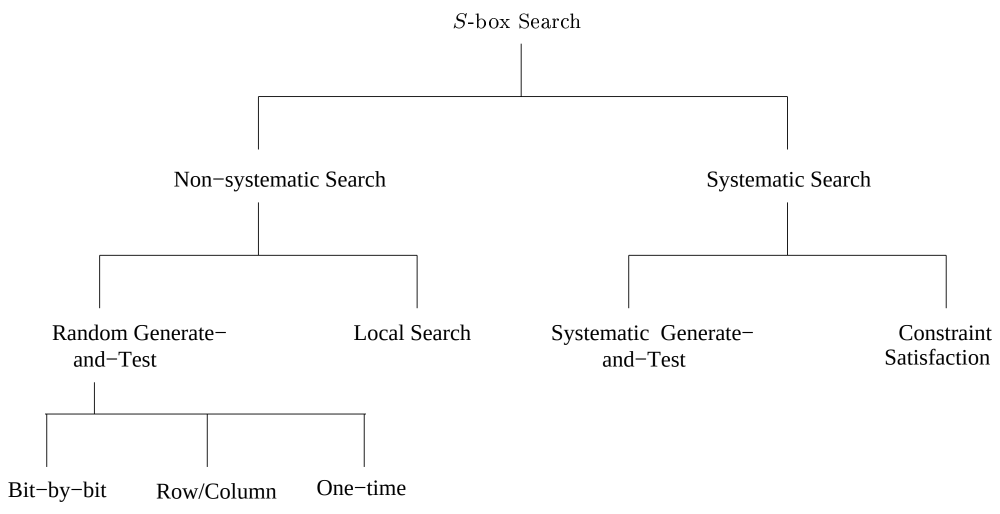


Figure 2.4: Classification of S-box Search Techniques

2.5.1 Nonsystematic Search

In this form of search, there is no systematic way in which S-boxes are searched from beginning to end. Two schemes immediately fall in this category: Random generate-and-test, and Local Search. The S-box search in the literature easily falls into one of these two categories.

Random generate-and-test

In this scheme, an S-box is filled with random entries and tested to satisfy the desired properties. Note that this class is akin to the one discussed in subsection 2.3.5. A striking feature summarized from the literature is the manner in which the S-box entries are loaded prior to testing. The approaches in the available literature, for our purposes, can be classified into three categories:

1. **Bit-by-bit generation-and-testing** In this approach, each bit of every S-box entry is loaded with a Boolean function and tested for satisfaction of the mathematical properties of the S-box properties up to the accumulation of the current bit. [38] loads a Bent function bit-by-bit, with the bits distributed across a column of an S-box, and tests for criteria such as its nonlinearity and highest dynamic distance. Individual bits of a 4×4 S-box entry are chosen and together, they are tested against the four design criteria such as bijection, nonlinearity, strict avalanche and output bit independence [72, 6,

5]. A recent approach uses neural networks to model Bent functions [31] since the sigmoid function used there lends itself naturally towards implementation of Bent functions. These can be loaded bit-by-bit into an S-box and tested for the other criteria. For our purposes, this approach clearly falls under the bit-by-bit generation-and-testing category of S-box search.

O'Connor [43] combinatorially analyzes this bit-by-bit scheme for $n \times n$ S-boxes and shows that there are practical limits up to which this approach can generate S-boxes efficiently. In particular, the author shows that the bit-by-bit method of generation can become infeasible when $m > 6$.

A cellular automata based approach to S-box design [63] employs one cell per bit of an S-box. The objectives are to maximize nonlinearity and minimize a second condition, that of *autocorrelation* of the S-boxes.

2. **Row / Column generation-and-testing** In this approach, each row of an S-box is generated and tested. The rationale behind this is that each row of an $n \times m$ S-box is a bijection, just as is the case with DES. Each entry of DES *in one row* would be viewed as a 4×4 bijective S-box now. The interactions between S-box entries as prescribed by the DES criteria **S-3** through **S-7** [16] is perceived in [6] as interactions *between* these 4×4 S-boxes, and is not addressed in the latter paper. The authors demonstrate that the S-boxes constructed using this approach proved superior to the $n \times n$ S-boxes constructed by Pieprzyk and Finkelstein [45] using the bit-by-bit approach. But that was not due to any problem in that approach. The limitation of this paper is that the authors focussed *only* on nonlinearity of S-boxes [6], and a number of other properties were not met.

3. **One-time generation of all entries, and testing of each** In this approach, an S-box is filled with entries and tests are carried out to check if the S-box satisfies mathematical properties. The paper by Cheng, et.al [73] devises a scheme by which they compute 8×8 S-boxes by encrypting the plaintext numbers $\{0, 1, \ldots, 255\}$ using randomly-generated 2-bit subkeys, and with a 6-round mini-version of the IDEA cipher. These S-boxes were tested for the properties of bijection, nonlinearity, avalanche criteria, output bit independence criteria, equiprobable input / output exclusive-OR distribution and in addition, the inverse S-boxes were also tested to satisfy these properties.

Local Search

In this approach, search for S-boxes is done using hill-climbing, genetic algorithms, and simulated annealing. Such meta-heuristic and evolutionary techniques have emerged as potentially very powerful tools for the design of S-boxes [14]. One feature of these techniques is the existence of a *cost function* that should be minimized or maximized. In [15], the cost function is based on the Walsh-Hadamard spectra. The requirement is that either the non-linearity should be high or the autocorrelation should be low [15], [13].

As part of S-box generation using this approach, an S-box is generated (randomly or otherwise) and then tested. This conforms to the one-time generate-and-test approach discussed in Section 2.5.1, except that the generation need not always be based on mathematical principles and could be random. After each S-box is filled with entries, one tries to perform a local search to optimize the cost function.

Millan [37] performed hill-climbing as part of the local search. Clark, Jacob and Stepney [15] performed a two-step local search. The first step is annealing to minimize the cost function. Let S_{sa} be the best S-box just encountered in the search process. The next step is to hill-climb from this point with respect to non-linearity, or with respect to autocorrelation, to produce the final solution f_{sahc} [15]. Then the non-linearity, autocorrelation and algebraic degree of f_{sahc} is measured. A comparison with Millan's method suggested that this was superior, and the authors infer that the hill-climbing step did not contribute significantly but the annealing step added to a dramatic increase in non-linearity.

The paper by Chakraborty, et.al [13] discusses experiments to determine the empirical values of two adjustment parameters X and R used in the spectrum-based cost functions. They have concluded that for an $n \times n$ bijective S-box, $R > 3.5$ and $X < 2^{\frac{n}{2}}$, and that R should be an integer.

Clark, Jacob, Maitra and Stanica [14] used a similar approach to search for Boolean functions, but a completely different search space. Instead of searching the space of Boolean functions for those with the desired properties of non-linearity, autocorrelation, among others, they searched the entire spectrum of artifacts (permutations of Walsh Transforms of all functions) to determine which of those are Boolean by spectral inversions (inverse Walsh Transforms). Annealing is again used as part of the search process. The cost function is the distance from a nearest Boolean function to which a (non-Boolean) element of the search space can "collapse" to. Using this idea, they have uncovered S-boxes that have hitherto been not obtained using any other means.

2.5.2 Systematic search

The advantage of local search over systematic search is that the memory requirements are small, and a solution can be found in a very short time [54]. For, one instantiates an S-box not satisfying the desired properties, and the guided search quickly homes to a near-neighbor that satisfies these properties.

The disadvantage is that, there is the extra effort of testing the assigned element of the search-space for those properties is exercised. Moreover, it is possible that the cost function gets optimized but the solution is still approximate. For example, [14] would give rise to *almost* Boolean functions.

We consider two approaches to systematic search: Generate-and-test and Constraint Satisfaction Problems (CSPs). Our work uses the CSP approach.

Systematic Generate-and-test

In this approach to systematic search, we assign values to each S-box entry starting from the lowest, and verify if all properties of the S-box are satisfied. If even one property is not satisfied, we discard that S-box and take the next value of the last-assigned variable. If all values of the last-assigned variable are exhausted, we backtrack and choose the next value for the penultimate variable, and so on.

This approach is very inefficient, particularly for large-size S-boxes. For a (n,n) bijective S-box, the worst-case number of searches is equal to $(2^n)^{2^n}$. Even for small-size S-boxes such as (4,4), the maximum number of checks is 16^{16}, i.e. 2^{64} (very high!). Hence this approach is not at all recommended.

Constraint Satisfaction Problem (CSP) Based Approach

This is the approach used in our work. The closest work of modeling security requirements using CSPs was presented by Bistarelli, et.al [8] to analyze security protocols. There the authors model the network that arises out of the execution of security protocols as a *Soft* CSP (SCSP) using the framework of semirings. The aspect of confidentiality, one of the goals of the security protocols, analyzed, is further formalized as the property of the solution of the SCSP. Two SCSPs have been posed: A policy SCSP that models the network arising out of protocol execution for those admissible protocols that have terminated successfully, and an *imputable* SCSP that models a given network configuration. The authors compare the solutions obtained for these two problems to determine whether the given configuration hides a confidentiality *attack*. The approach is demonstrated on the Needham-Scroeder Security Protocol based on Asymmetric Key Cryptosystems.

To our knowledge, employing CSPs is a first-time approach to S-box design. We generate an entire S-box already satisfying all of the properties that are modeled as constraints. The S-boxes are generated using propagation and inferencing. No extra step of testing is required as is being done in the earlier approaches, since, the constraints are satisfied as part of solution generation, suggesting a major advantage of the CSP approach over the existing ones. Another advantage using this approach is that each S-box entry is complete in itself unlike those in the literature where each S-box is populated a bit at a time, or a row/column at a time. We will introduce the novel idea of a *Partially Assigned S-box* in Chapter 3, in which an S-box does not have all entries assigned. This assignment can be done either row-wise or column-wise depending upon the heuristic used. The partially assigned S-box will incrementally be extended to a complete S-box with all entries assigned, and all requirements satisfied at any point in search. The idea of extending a partially assigned S-box gives rise to a set of incremental heuristics discussed in Chapters 4 and 5, to significantly speed up S-box search as the experimental results in Chapter 7 will reveal.

2.6 Constraint Satisfaction Problems (CSPs)

A *Constraint Satisfaction Problem* (CSP) is a triplet (X, D, C) where:

1. X is a set of *variables*, $X = \{x_1, x_2, \ldots, x_n\}$
2. D is a set of *domains*, $D = \{D_1, D_2, \ldots, D_n\}$. Each of the D_i's is in itself a set of domain-values that the variable x_i, $1 \leq i \leq n$, can assume.
3. C is a set of relations on a subset of the set X of variables. Each element of C, say C_i, for some i, is a relation R_i defined on a subset $S_i \subseteq X$, denoting valid assignments to the variables in S_i simultaneously. If $S_i = \{x_{i1}, x_{i2}, \ldots, x_{ir}\}$ is the set of variables with variable x_{i_k} having domain D_{i_k}, $1 \leq k \leq r$, then R_i is a subset of the Cartesian Product $D_{i1} \times D_{i2} \times D_{i3} \times \ldots \times D_{ir}$. A constraint C_i on the variables in S_i can also be written as a pair $C_i = \langle S_i, R_i \rangle$.

Definition 2.1 (Binary and n-ary Constraint) *If a relation $c \in C$ on a subset of the set X of variables is binary, then the particular constraint in the set C is called a* binary *constraint, having two variables. A constraint with more than two variables is called an n-*ary constraint.

Definition 2.2 (Instantiation) *An instantiation of a set of variables $\{x_{i_1}, x_{i_2}, \ldots, x_{i_r}\} \subseteq X$ where each variable x_{i_k} has domain D_{i_k}, $1 \leq k \leq r$, is a tuple of ordered pairs $\langle (x_{i_1}, a_{i_1}), (x_{i_2}, a_{i_2}), \ldots, (x_{i_k}, a_{i_k}) \rangle$ in which each ordered pair (x_{i_k}, a_{i_k}) represents an assignment of the value $a_{i_k} \in D_{i_k}$ to the variable x_{i_k}, $1 \leq k \leq r$ [20].*

Informally, an instantiation of a subset of the set of variables is an assignment of an element to each variable in the set from its domain. The tuple is alternatively written as $(x_1 = a_1, x_2 = a_2, \ldots, x_r = a_r)$, or even more compactly, as $(a_1, a_2, \ldots, a_r)$. In Chapters 4 and 5, we will discuss the novel idea of a *partially assigned S-box*, which is an instantiation of a subset of the set of variables in our model.

Definition 2.3 (Solution to a CSP) *A solution of a CSP (X, D, C) is an instantiation of all its variables in X such that all constraints in C is satisfied.*

The S-boxes in our work that have all entries present such that all security criteria are satisfied form the solutions to the CSP.

2.6.1 Representation of a CSP

For our purposes, we consider three ways to represent a CSP [20]. The first of these is used in our work while the other two are mentioned for completeness purposes.

Boolean Matrix Consider a binary constraint with variables x and y. Let their domains be $D_x = \{d_0, d_1, d_2, \ldots, d_{m-1}\}$ and $D_y = \{e_0, e_1, e_2, \ldots, e_{n-1}\}$, respectively. A binary constraint involving x and y can be represented as a Boolean $m \times n$ matrix $[a_{ij}]_{m \times n}$. In this representation, the binary element a_{ij}, $0 \leq i < m$, $0 \leq j < n$, is defined as follows:

$$a_{ij} = \begin{cases} 1, & \text{if } x = d_i \wedge y = e_j \\ 0, & \text{otherwise} \end{cases}$$

The Boolean matrix representation is also referred to as the *extensional representation* or *extensional form*, and is the representation used in our work.

We now study this representation with the help of three examples.

Example 2.1 *Consider a CSP with two variables x and y, and domains D_x and D_y respectively. Let $D_x = D_y = \{0, 1, 2, 3\}$. Consider the following constraint involving x and y:*

$$x + y = 4 \tag{2.4}$$

Constraint 2.4 can be written in an expanded form as follows, consistent with D_x and D_y:

$$x = 1 \wedge y = 3,\ x = 2 \wedge y = 2,\ x = 3 \wedge y = 1$$

From this expansion, the extensional representation of constraint 2.4 results in the following Boolean matrix A, with the elements of D_x forming its row numbers and the elements of D_y, its column numbers. In other words, the row and column numbering for the matrix A begin from 0 rather than 1. Thus an entry in the matrix A in Row 1 and Column 3 is 1, corresponding to the Boolean expression $x = 1 \wedge y = 3$ *and similarly, for the other two Boolean expressions.*

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

This particular constraint results in a square, symmetric matrix but in general, the matrix need neither be square nor symmetric.

When there exists more than one binary constraint involving the *same* two variables, they can be combined by AND-ing the Boolean entries in their extensional representation. Example 2.2 discusses this property.

Example 2.2 *Consider a CSP with two variables x and y, and domains D_x and D_y respectively. Let $D_x = D_y = \{0, 1, 2, 3\}$. Consider the following binary constraints involving the same two variables x and y:*

$$\begin{aligned} x + y &= 4 \\ x - y &= 2 \end{aligned} \tag{2.5}$$

Constraints 2.5 can be expanded into the following:

$$x = 1 \wedge y = 3,\ x = 2 \wedge y = 2,\ x = 3 \wedge y = 1$$
$$x = 2 \wedge y = 0,\ x = 3 \wedge y = 1$$

We now have two expanded constraints in x and y. Note that both constraints should be satisfied simultaneously. The expanded constraints are represented in extentional form using matrices A_1 and A_2, respectively, to yield:

$$A_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix},\ A_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

By ANDing the entries in the two matrices A_1 and A_2, we obtain the following matrix:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{2.6}$$

The AND-operation is the composition used since both constraints 2.5 should hold simultaneously. This what we will do in our S-box formulation in Chapter 3.

The constraint resulting from the matrix A that serves as its extensional representation (Equation 2.6) is

$$x = 3 \wedge y = 1$$

which is none other than the solution to the CSP.

The following example demonstrates the effect of reordering the domains of variables on the extensional representation of a constraint.

Example 2.3 *Consider a CSP with two variables x and y, and domains D_x and D_y respectively. Let $D_x = D_y = \{0, 1, 2, 3\}$. Consider the following constraint involving x and y:*

$$x + 2y = 4 \tag{2.7}$$

Equation 2.7 can be expanded into the following:

$$x = 0 \wedge y = 2, \; x = 2 \wedge y = 1 \tag{2.8}$$

The extensional representation of constraints 2.8 is given by the following 4×4 matrix.

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Now let us order the domains D_x and D_y to yield ordered domains $D'_x = \{2, 0, 3, 1\}$ and $D'_y = \{3, 2, 1, 0\}$.

To maintain Eq. 2.8, the rows and columns of the matrix A should now be shuffled. Shuffling the entries accordingly results in the following matrix A':

$$A' = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.9}$$

Define permutation functions $\lambda_x : D_x \rightarrow D'_x$ and $\lambda_y : D_y \rightarrow D'_y$, where:

$$\lambda_x = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 3 & 1 \end{pmatrix}, \ \lambda_y = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$

Let us now attempt to determine the matrix A' directly from permutations λ_x and λ_y. Applying λ_x and λ_y directly on the numbers involved in constraint 2.8 will not yield A' as the extensional representation of the resulting constraint. For, direct application yields the following:

$$x = \lambda_x(0) \wedge y = \lambda_y(2), x = \lambda_x(2) \wedge y = \lambda_y(1)$$

or

$$x = 2 \wedge y = 1, x = 3 \wedge y = 2$$

resulting in the extensional representation

$$A'' = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \neq A'$$

The inverse permutations of λ_x and λ_y are given by:

$$\lambda_x^{-1} = \begin{pmatrix} 2 & 0 & 3 & 1 \\ 0 & 1 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 3 & 0 & 2 \end{pmatrix}$$

and

$$\lambda_y^{-1} = \begin{pmatrix} 3 & 2 & 1 & 0 \\ 0 & 1 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$

It turns out that application of the <u>inverses</u> λ_x^{-1} and λ_y^{-1} to the numbers in constraint 2.8 yields the desired result. This application implies the following:

$$x = \lambda_x^{-1}(0) \wedge y = \lambda_y^{-1}(2), x = \lambda_x^{-1}(2) \wedge y = \lambda_y^{-1}(1)$$

or

$$x = 1 \wedge y = 1, x = 0 \wedge y = 2$$

which results in the matrix A' given by Equation 2.9.

Constraint Graph A binary constraint can be graphically represented by a constraint graph. The variables of the set X serve as the nodes of the graph. Whenever any two variables participate in a particular constraint, they are connected by an edge. A dual graph is obtained by interchanging the original graph's nodes into edges and edges into nodes.

Hypergraph For n-ary constraints, all variables participating in any one constraint are lumped into a node of the graph. Thus each node is a subset of X. Two nodes, formed by subsets X_1 and X_2, are connected if and only if the set $X_1 \cap X_2$ is non-empty. Such a graph is called a *hypergraph*. An n-ary constraint is converted into a set of binary constraints using the dual graph of its underlying hypergraph.

2.6.2 An overview of S-box Search Strategy using CSPs

To examine the feasibility of the CSP approach to S-box search, we modeled the eight criteria of DES into constraints. Criterion **S-8** is found to deal with multiple S-boxes and is handled separately at another level, discussed in Chapter 6. Criterion **S-1** to **S-7** are modeled as constraints. Out of these, criterion **S-1** will be found to be already modeled based on variable choice. **S-3** to **S-6** are binary constraints while **S-2** and **S-7** are n-ary constraints.

Initially we tried to model the constraints using the CSP programming language Mozart-Oz [67]. However we quickly found that a number of DES constraints had to be checked at the *bit* level, for which Mozart-Oz proved to be insufficient. Therefore, we resorted to a C^{++}—based solver to take in our model in its appropriate form, and generate solutions.

The solver is introduced in Section 2.7 in pseudo-code form (Algorithm 1). In its original form, the solver handles only binary constraints. Our strategy is to model constraints for criteria **S-3** to **S-6** (Chapter 3) and precompile these into the solver, generate solutions to satisfy these binary constraints, and then test them on constraints S-2 and S-7. The approach turned out to be very inefficient (systematic generate-and-test!) In general, generate-and-test is not recommended as discussed earlier, and *constraint propagation* needs to be carried out. We formulate several heuristics for the constraints that model criteria **S-2** (Chapter 4 and **S-7** (Chapter 5).

For the binary constraints, we discuss constraint propagation that is immediately related to our work, namely, arc-consistency, and its relaxed forms, viz. bounds consistency and range consistency. `Alldiff` constraints will be addressed next, particularly suited for **S-3**.

2.6.3 An Overview of Constraint Propagation

Old CSP solvers employed so called *intelligent backtrackers*, namely algorithms for intelligent management of conflict recording between values of variables. Some examples of intelligent backtrackers are: graph-based backjumping, conflict-based backjumping, dynamic backtracking and backmarking [20, 47, 22, 29]. These algorithms are able to efficiently identify relevant culprit variables for found conflicts, and thereby avoid some redundant search.

Modern solvers use look-ahead techniques, such as: forward checking, arc and path consistency, or singleton consistency [56, 39]. A scheme that, based on (parts of) a CSP P, infers additional (redundant) constraints to be added to P, is called a filtering operation. Given a filtering operation $\mathcal{A}$ for CSPs,

$$\mathcal{A} : CSP \rightarrow CSP$$

a CSP P is locally consistent with respect to $\mathcal{A}$ (e.g., arc consistent, path consistent, etc.) if it is a fixed point of $\mathcal{A}$, i.e.,

$$P = \mathcal{A}(P).$$

One of the most commonly used local consistency concepts is *arc consistency.* It is based on the Waltz filtering operation [71] that uses a constraint C_k on two variables x_1 and x_2 subject to unary constraints $x_1 \in D_1$ and $x_2 \in D_2$ respectively. This filtering operation infers the new constraints with stronger D_1 and D_2:

$$\begin{aligned} D_1 &:= D_1 \cap C_k(D_2)|_{x_1} \\ D_2 &:= D_2 \cap C_k(D_1)|_{x_2}, \end{aligned}$$

where by $C_k(D_j)|_{x_i}$ we denote the projection of the domain D_j for x_j, through the relation C_k, on the variable x_i. Practically, Waltz filtering removes values from the domains of variables. New domains (sets in unary constraints) obtained this way are traditionally referred to as *labels* of the corresponding variables. The repeated application of the Waltz filtering operation until reaching the fixed point is called arc consistency (AC). The labels obtained at fixed point by repeated application of this filtering process are called *arc consistent labels.* The CSP whose domains are arc consistent labels is said to be arc consistent. If arc consistency removes all the elements from the domain of some variable (also called *domain wipe-out*), we can infer that the CSP has no solution.

Various ways to repeatedly apply Waltz filtering have been studied in the past. They differ by the way of selecting the constraint for the next filtering operation, or by the additional information stored about the progress of the last filtering step on each given constraint. Such information can help in only incrementally filtering the constraint when the labels of its variables change. The best such filtering algorithms are AC2001 and AC-3$_d$ [9, 66]. Sometimes filtering algorithms are not applied repeatedly until the fixed point is achieved. For example, they may be applied only a fixed number of times, as in the case of *directed arc consistency* [68].

Look-ahead (local filtering) can be either used in a pre-processing step, or repeatedly called at various points throughout the search process. The latter case is referred to as maintenance of the corresponding consistency concept. Maintenance of arc consistency (MAC) [55] is commonly referred to as one of the best

CSP solvers. Each assignment of a value to a variable (called the current variable) is propagated as if a filtering step would have restricted the label of that variable to the corresponding value. MAC has been studied under several versions, differentiated by the selection of the points at which the arc consistency is enforced. Typically it is enforced after any change of a label.

2.6.4 Forms of Arc-Consistency of CSPs

Arc-consistency is extended to Generalized Arc Consistency (GAC), and also, is slightly relaxed as Bounds Consistency and Range Consistency. Bounds and Range consistency are used in `Alldiff` constraints, to be discussed next. We also mention *Singleton* consistency which is used in the C^{++} solver employed in our work.

Generalized Arc Consistency (GAC)

While arc-consistency is applicable to binary constraints, this notion is extended to the domains of more than two variables participating in an n-ary constraint.

Dechter [20] defines GAC as follows: Given a constraint network $R = (X, D, C)$ with $S \subseteq X$ and $R_S \in C$, a variable x is arc-consistent relative to R_S if and only if, for every value $a \in D_x$ there exists a tuple $t \in S$ in the domain of variables in S, such that $t[x] = a$. t can be called a *support* for a.

Dechter discusses how AC-1 can be extended to GAC and informs that the complexity of the main step in GAC is bounded by $O(d^{r+1})$, where d bounds the domain-size and r is the constraint scope size. Bessiere, et.al. [10] extended AC-2001/3.1 to GAC2001/3.1. The complexity of GAC2001/3.1 is $O(er^2d^r)$, where e is the number of constraints.

In the next two definitions below, 1,5 form a two-element set while [1,5] is an *interval* (actually, a closed interval), i.e. the set $\{1, 2, 3, 4, 5\}$.

Range Consistency

An n-ary constraint $C(x_1, x_2, \ldots, x_n)$ where no domain D_i is empty, is called *range*-consistent [27] if and only if, for each variable x_i and value $d_i \in D_i$, there exist values $d_1 \in [\min D_1, \max D_1]$, $d_2 \in [\min D_2, \max D_2]$, …, $d_{i-1} \in [\min D_{i-1}, \max D_{i-1}]$, $d_{i+1}, \in [\min D_{i+1}, \max D_{i+1}]$, …, $d_n \in [\min D_n, \max D_n]$ such that $(d_1, d_2, \ldots, d_n) \in C$.

This means that range consistency does not check for the feasibility of the constraint for each and every domain-value of the participating variables, but only with respect to intervals that include those domain-values.

Bounds Consistency

An n-ary constraint $C(x_1, x_2, \ldots, x_n)$ where no domain D_i is empty, is called *bounds*-consistent [27] if and only if, for each variable x_i and value $d_i \in \{\min D_i, \max D_i\}$, there exist values $d_1 \in [\min D_1, \max D_1]$, $d_2 \in [\min D_2, \max D_2]$, …, $d_{i-1} \in [\min D_{i-1}, \max D_{i-1}]$, $d_{i+1}, \in [\min D_{i+1}, \max D_{i+1}]$, …, $d_n \in [\min D_n, \max D_n]$ such that $(d_1, d_2, \ldots, d_n) \in C$.

This means that bounds consistency does not even check all domain-values of the variables participating in the constraint, but only the minimum and maximum values. Bounds consistency can be viewed as a relaxation of range consistency.

`Alldiff` *Constraints*

While formulating constraints for our CSP solver for DES S-boxes, the criterion S-2 is found to be made of n-ary constraints, specifying that each S-box row should not contain duplicates. In other words, the variables participating in each row should possess different values. Such constraints can be easily modeled as `Alldiff` constraints and consistency algorithms specially available for these kinds of constraint can be used to advantage [27], [49], [51], [50].

The programming language Mozart-Oz [67] provides an `ALLDISTINCT` module that helps the developer to specify an `Alldiff` constraint. We were able to use this module to advantage for generating solutions that satisfied criterion S-2 alone.

Simply decomposing an `Alldiff` constraint having n variables into ${}^nC_2$ binary constraints does not always give us the desired performance. In fact, the pruning performance is poor, with a complexity of $O(n^2)$ [27]. In comparison, the original set of `Alldiff` constrains performs at $O(dn^{1.5})$, where d is the maximum cardinality of domains.

A naiv̈e algorithm that is $O(n^3)$ in the number of variables is discussed in [27]. This algorithm considers an interval $I = [a, b]$ where a is the smallest value of all domains and b, the largest. If $\#I$ is less than the cardinality of the variables participating in I, there is no solution. If I is a Hall Interval, the bounds are updated. Regin's algorithm [50], is an improvement over the naiv̈e implementation and has $O(n^2d^2)$, where d is the maximum domain-size. Mehlhorn and Thiel present another that performs in time $O(n)$ plus the time required for sorting the endpoints of the intervals.

Puget's algorithm [49] for bounds consistency of `Alldiff` constraints is $O(n \log_2 n)$.

Leconte's range consistency algorithm runs in time $O(n^2d)$, where d is the average domain size.

Régin's hyper-arc consistency algorithm on `Alldiff`, based on matching theory, constructs a value graph in time $O(d|X_C| + |X_C| + |D_C|)$ where the subscripted

C's stand for the variables participating in the matching cover of the value graph. d is the maximum cardinality of the domains of the variables. Hopcroft and Karp implemented a maximum matching algorithm on the value graph, that runs in time $O(\sqrt{|X_C|}m)$, where m is the number of edges in the value graph.

Régin proposes another form of `Alldiff`, namely, the symmetric `Alldiff` constraint [51]. This is equivalently expressible as an `Alldiff` constraint along with additional information on symmetry. However the symmetric form exhibits more global information that the split-version of the CSP. The additional information is used for pruning.

2.6.5 Limited Discrepancy Search

While running our CSP solver, we often obtain several solutions that appear identical in the values assigned to the first few variables. We may want to limit such solutions and move on to those that look “different".

Tree search methods are useful for solving many practical problems because carefully-tuned successor-ordering heuristics guide the search towards regions of the space that are likely to contain solutions.

Limited Discrepancy Search (LDS) is introduced by Harvey and Ginsberg [24]. During search, when a goal node is not reached, the thinking is that the search would have succeeded had it not been for a small number of “wrong turns" along the way. The point at which the wrong turn occurs is referred to as a discrepancy. This paper demonstrates a novel search technique called Limited Discrepancy Search, which is a backtracking algorithm that searches the nodes of the tree in increasing order of such discrepancies. The paper shows how this technique can be expected to outperform existing approaches.

The existing approaches discussed are iterative sampling and backtracking. In iterative sampling, without successor-ordering heuristic, iterative sampling is ineffective when the solution density is not very high. In backtracking, mistakes made early in the search process, particularly when the subtree is large or when there are very few solutions present, present a tremendous burden on the heuristic early in the search process.

The idea of LDS is that when a heuristic fails, it would have led to a solution if only it had not made one or two “wrong turns" that got it off track. One should be able to systematically follow the heuristic at all but one discrepancy. If that fails, one can follow the heuristic at all but two discrepancies. And so on. Thus, LDS does a depth-first search traversal of the tree, limiting the number of discrepancies to a discrepancy limit x. When, eventually, $x = d$, where d is the maximum depth of the tree, then the search becomes exhaustive.

A theoretical comparison is made with the two existing methods, examining the likelihood of finding a solution in the specified amount of time. The paper defines a *mistake probability* m as the probability that a randomly selected child of a good node is not having any goals in its subtree. The heuristic probability p is equal to $(1 - m)$ (or greater, if it does better than random selection). The chance of finding a solution on a random path (iterative sampling) to depth d is $(1-m)^d$. Given specific values for p and m, the paper demonstrates the theoretical probabilities of success as a function of the height of the tree for iterative sampling, chronological backtracking and LDS for various values of p.

When $p = 0.8$, LDS performs slightly worse under these conditions. As p increases to 0.85, 0.9, 0.95, the curve for LDS goes way higher compared to the others, suggesting that the performance of LDS increases dramatically with better p.

Richard [30] discusses an improvement of Limited Discrepancy Search over the earlier one by Harvey and Ginsberg [24]. In the improved version (ILDS), a number of search paths that were repeatedly traversed in each iteration in the earlier version (OLDS) are eliminated.

An analytical comparison of the two algorithms is made in the paper. If the depth of the search tree is d, then the total number of paths generated by OLDS in a complete search to depth d is $(d + 2)2^{d-1}$. This is also the asymptotic time complexity of OLDS. In comparison, the complexity of ILDS is $O(2^d)$, suggesting that OLDS may be worse than ILDS by a factor of $\frac{d+2}{2}$, in the worst-case scenario. Usually this does not happen since, OLDS was designed for very large trees and only a few iterations were needed. Another reason is that when integrated into CSPs or branch-and-bound methods, a great deal of pruning disallows reaching all the way down to the maximum depth.

The paper analytically compares ILDS with DFS since the authors found that ILDS is still inefficient to DFS due to the larger number of internal nodes generated by ILDS. If b is the branching factor and d, the depth of the tree, then for DFS, the total number of nodes is $\frac{b^{d+1}-1}{b-1}$. In comparison, for ILDS, the total number of nodes is found to be equal to $\frac{b(b^{d+1}-1)}{(b-1)^2}$. The ratio of the total number of nodes generated in ILDS to that in DFS is equal to $\frac{b}{b-1}$. The expressions were approximated in the paper but that was not necessary; the ratio is still the same. As the branching factor increases, the ratio decreases. When pruning happens, the overhead of ILDS increases.

Ineffectiveness of Limited Discrepancy Search for our work

We discovered that limited discrepancy search is not effective for our work. LDS is used where a goal assignment is not found, to expand in a different direction using the notion of the “number of wrong turns". However, in our work, we already have several goal assignments. Our objective is to see “different" kinds of goal assignments rather than the “same kinds" of goal assignments. LDS will certainly give us “different" kinds of goal assignments upon expanding in a different direction. But, it will still give us the “same" kind of assignments within that “different" kind. As such, this approach is not recommended, and we consider reordering the domains according to a random permutation.

2.7 The Solver for Binary Constraints

Algorithm 1 outlines the solver in its original form. The solver handles binary constraints represented in extensional form, pre-computed and input to it. The solver generates S-boxes employing Maintenance of Arc Consistency (MAC) [55] with AC2001 [9].

Procedure `Solver` is recursive. The inputs are the set of variables X, the set of (reduced) domains D', the precomputed binary constraints C_2, recursion level k and threshold τ of optimality of the solutions sought. The solver is invoked for the first time with $D' = D$ (the initial set of domains), and $k = 0$ (to indicate the starting variable). Here the function `ProcessNary`() returns **true** if $k = |X|$. This is the condition that all variables are assigned and forms the base case for recursion to terminate. Function `MakeAndCheckSBox`() creates an S-box Φ with all values assigned, and returns the same. Functions `InitPartialVarsAndCounts`(), `ReverseUpdateCountDistSet`() and `ReverseUpdateCount`() perform no operation. Boolean functions `CheckPartialSBox`() and `ProcessOtherDomains`() simply return **true**. Under these circumstances, when all variables are assigned, Line 9 prints the solution that satisfies all binary constraints in C_2. All of the functions that hitherto return **true** or perform trivial operations (or even none) implement the heuristics proposed in the paper for the global n-ary constraints for **S-2** and **S-7**.

Procedure `SelectNextVariable`, called in Line 3 selects the next variable X_j and returns its index j, as governed by a permutation $\pi : \mathbb{Z}_{|X|} \rightarrow \mathbb{Z}_{|X|}$. Two variable-ordering heuristics are considered in our work for performance evaluations (subsection 5.5.2), namely, the default straight-line and an alternative, zig-zag ordering.

In Line 11, the next available value v is assigned to $x_j \in X$ from its reduced domain D'_j. All other elements in D'_j are added to a deletion set DS (Line 12). Establishment of Arc Consistency using AC2001 is made at Line 16 by the function `EstablishAC`, filtering domains in D' to return a set D'' of reduced-domains. If no reduced-domain in D'' is empty, the solver recurses in Line 17. The function

Procedure Solver(X, D', C_2, k, τ)

```
input : Variables $X = \{x_0, x_1, \ldots, x_{2^n-1}\}$,
        Domain-subsets $D' = \{D'_i : 0 \leq i < 2^n\}$, with $D'_k \subseteq D_k$
        being the domain-subset for $x_k$,
        $C_2$ is the set of binary constraints for the 4 criteria **S-3** to **S-6**,
        $k$ is the recursion level ($x_k \in X$ is the current variable),
        $\tau$ = The threshold score sought for each $S$-box solution.
output: $n \times m$ $S$-boxes having a maximum score equal to $\tau$
begin
    InitPartialVarsAndCounts()
    $j \leftarrow$ SelectNextVariable($k$)
    $c_1 \leftarrow$ CheckPartialSBox()
    if $c_1$ then
        $c_2 \leftarrow$ ProcessNary()
        if $c_2$ then
            $\Phi \leftarrow$ MakeAndCheckSBox()
            PrintSBox($\Phi$)
        else
            foreach $v \in D'_j$ do
                $DS \leftarrow \{(j, w) : w \in D'_j \wedge w \neq v\}$
                $D'_j \leftarrow \{v\}$
                $c_4 \leftarrow$ ProcessOtherDomains($DS, D', j, v$)
                if $c_4$ then
                    $D'' \leftarrow$ EstablishAC($D', DS$)
                    if *no domain in $D''$ is empty* then
                        Solver($X, D'', C_2, k+1, \tau$)
                ReverseUpdateCountDistSet()
                $D'_j \leftarrow$ RestoreDomain($x_j$)
        ReverseUpdateCount()
```

Procedure SelectNextVariable(k)

```
begin
    return $\pi(k)$ ;
```

`RestoreDomain`() restores the domain D'_j of x_j before the next value from D'_j is considered at Line 11.

2.8 Notations

For a number x, we use $|x|$ to denote its absolute value. If X is a set, then $|X|$ represents its cardinality (number of elements in the set X). Whenever a set is written with braces, its cardinality is written with a # preceding the set itself. For example, the cardinality of the set $\{a_0, a_1, a_2, \ldots a_{k-1}\}$ is written as $\#\{a_0, a_1, a_2, \ldots a_{k-1}\}$.

The symbol $\oplus$ denotes the bitwise exclusive-OR (or XOR) of two bit patterns a and b having identical bitlength, and the operation is written as $a \oplus b$. The symbol $\cdot$ denotes the bitwise AND of two quantities a and b having identical bitlength and the operation is written as $a \cdot b$. The one's-complement of a bit pattern a, also called the negation of a or the NOT-operation on a, is written as $\bar{a}$. In logical expressions (as opposed to bitwise arithmetic), the symbol $\wedge$ is used for conjunction, the symbol $\vee$ used for disjunction and the symbol $\neg$, used for negation.

A linear combination of Boolean variables $x_0, x_1, x_2, \ldots, x_{k-1}$, is given by the expression

$$\bigoplus_{i=0}^{k-1} a_i \cdot x_i = a_0 \cdot x_0 \oplus \ldots \oplus a_{k-1} \cdot x_{k-1}$$

where a_i are Boolean coefficients, $\oplus$ is the bit-wise exclusive-OR operator and $\cdot$ the bit-wise AND operator. A linear Boolean function $L_\omega(x)$ on an n-bit input $x = x_0 \ldots x_{n-1}$ defined by an n-bit selector $\omega = \omega_0 \ldots \omega_{n-1}$ is computed [14] as:

$$L_\omega(x) = \omega_0 \cdot x_0 \oplus \ldots \oplus \omega_{n-1} \cdot x_{n-1} = \bigoplus_{i=0}^{n-1} \omega_i \cdot x_i \tag{2.10}$$

The *parity* of a binary quantity a is equal to the number of one's in a. If a has an odd number of 1's, it is said to follow *odd parity* and if this number is even, it is said to have *even parity*. The check for parity is made by computing the exclusive-OR of the bits in a. The result of the exclusive-OR is either 0 or 1; if 1, a possesses odd parity, otherwise a possesses even parity. It is easy to see that the parity of a is obtained by taking the sum of the bits in a modulo-2.

Some existing criteria are based on the concepts of *Hamming weight* and *Hamming Distance*. The *Hamming weight* of a given bit-pattern u, denoted by $wt(u)$, is defined as the number of 1's in u. The *difference* between two n-bit numbers x

and y is equal to $x \oplus y$. The *Hamming Distance* between x and y is the minimum number of changes to be made to x to obtain y, and is equal to $wt(x \oplus y)$.

Chapter 3
CSP Models for Security Criteria

We model the security criteria listed in Table 2.1 and discuss constraint formulation strategies. Binary constraints are formulated in this Chapter, and n-ary constraints, in Chapters 4 and 5 to model the security requirements.

3.1 Constraint Formulation Strategy

Some of the S-box criteria of Table 2.1 can be formulated as binary constraints while others get formulated as n-ary constraints. Moreover, some criteria may operate on a combination of solutions generated. A strategy is put in place before attempting to formulate constraints for each criterion.

3.1.1 One S-box, and combination of S-boxes

In the S-box criteria one can observe that **S-8** applies to more than one S-box taken together (three S-boxes in this case). All other criteria apply to only one S-box. The following two-step strategy for constraint formulation is adopted in this work.

1. Generate S-boxes that satisfy all criteria except **S-8**, filtering out all those S-boxes that do not satisfy these other criteria.

2. Consider a subset of the set of generated solutions, and repeatedly validate **S-8** on all such subsets. Only those solutions that satisfy all eight criteria remain behind while all others get filtered out.

The details of Step 1 are discussed throughout this Chapter and also, in Chapters 4 and 5. Step 2 is formulated as a separate problem and discussed in Chapter 6.

3.1.2 Binary and n-ary constraints of DES Criteria

In the S-box criteria of Table 2.1, one can further observe the following: **S-2** and **S-7** gives rise to n-ary constraints that take in more than two variables, while the remaining criteria **S-3** to **S-6** all give rise to binary constraints. In this work, the following steps are followed in the order below:

1. The remaining criteria **S-3** to **S-6** are first analyzed and are modeled as binary constraints in this Chapter.

2. **S-2** is positioned to validate and accept only those solutions satisfying it. In that sense **S-2** serves as a “filter". This is a generate-and-test approach. Decomposition of **S-2** into hard constraints by employing projection, resulting in heuristics for search speedup, are proposed in Chapter 4.

3. Those S-boxes accepted by **S-2** will be further filtered by a second “filter" that implements the n-ary global constraint **S-7**. This is also a generate-and-test approach. A incremental approach, and a novel technique of projecting on the domains of future variables, is discussed in Chapter 5 to speed up the constraint for **S-7**.

3.1.3 A problem solver

To implement Step 1 mentioned in subsection 3.1.1, the binary constraints formulated for criteria **S-3** to **S-6** will be precompiled into a solver that will emit outputs to satisfy only these criteria.

Now the solver will be modified to include heuristics for the n-ary constraint **S-2** and **S-7**. The solver will take in the partially-correct solutions and run the n-ary constraints to yield S-boxes that will satisfy all eight DES criteria.

3.2 The CSP Formulation

We will now discuss the variables, domains for each variable, and constraint formulation. **S-2** is referred to as the *nonlinearity* constraint and its formulation is dealt with in Chapter 4. We will refer to criterion **S-7** as the *COUNT* constraint for lack of a suitable terminology in the literature. The modeling of this criterion will be discussed in Chapter 5.

3.2.1 Variables of the CSP Model

For an $n \times m$ S-box requiring n inputs, 2^n variables are required. These variables will be represented by x_i, where each x_i is the output of one S-box, $0 \leq i \leq 2^n - 1$. The set of variables for the CSP is the set $\{x_0, x_1, \ldots, x_{2^n-1}\}$.

Specifically, for 6×4 S-boxes such as those used in DES, with $n = 6$, there are 64 values of the 4-bit output of an S-box, each output corresponding to one

6-bit input. Let these 64 values be represented by variables $x_0, x_1, \ldots x_{63}$, with $x_i \in \{x : 0 \leq x \leq 15\}$, $0 \leq i \leq 63$. Each variable x_i specifies the output of an S-box corresponding to input i of the S-box, $0 \leq i \leq 63$. For example, x_0 corresponds to 6-bit input 0 ($\mathbf{000000_2}$, in binary). Output x_1 corresponds to 6-bit input 1 ($\mathbf{000001_2}$ in binary), and so on.

The set X of the 64 variables for a 6×4 S-box is given by

$$X = \{x_0, x_1, x_2, \ldots x_{63}\}$$

Using these variables, an 6×4 S-box is organized as shown in Table 3.1. For convenience, the values for the two-bit numbers x_0x_5 (row index) and the four-bit numbers $x_1x_2x_3x_4$ (column index) of an input to the S-box are expressed in the decimal number system.

The peculiar pattern in which the 6×4 S-box entries x_i is organized is observed in Figure 3.1. For example, while traversing across the first row from left to right, the variables are listed as $x_0, x_2, \ldots x_{30}$ instead of the more intuitive listing $x_0, x_1, \ldots x_{15}$. Similarly in the second row, only the x's with odd subscripts appear. This interesting layout is due to the fact that the first and last bits of the six-bit input, and not the two leftmost bits, form the row-selection. (Equivalently, this is because the middle four and not the last four bits of the input to a 6×4 S-box forms the column-selection.)

	$x_1x_2x_3x_4$								
x_0x_5	0	1	2	3	$\ldots$	12	13	14	15
0	x_0	x_2	x_4	x_6	$\ldots$	x_{24}	x_{26}	x_{28}	x_{30}
1	x_1	x_3	x_5	x_7	$\ldots$	x_{25}	x_{27}	x_{29}	x_{31}
2	x_{32}	x_{34}	x_{36}	x_{38}	$\ldots$	x_{56}	x_{58}	x_{60}	x_{62}
3	x_{33}	x_{35}	x_{37}	x_{39}	$\ldots$	x_{57}	x_{59}	x_{61}	x_{63}

Table 3.1: Distribution of the constraint variables in a 6×4 S-box

3.2.2 Domain of each variable, and Domain-set of the CSP

For an $n \times m$ S-box that yields an m-bit output, each variable $x_i \in X, 0 \leq i \leq 2^m - 1$ assumes integral values from the set $\{0, 1, \ldots, (2^m - 1)\}$. The domain D_i for each x_i is then given by:

$$D_i = \{0, 1, 2, \ldots, 2^m - 1\}$$

The domain-set D of the CSP is the set of such domains, one for each variable.

A 6×4 S-box such as the one used in DES yields a 4-bit output value ($m = 4$) ranging from 0 up to a maximum of 15. In other words the domain of each variable x_i in the set $X = \{x_0, x_1, \ldots, x_{15}\}$ of variables, is the set $D_i, 0 \leq i \leq 63$, given by:

$$D_i = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$$

The domain-set D of the CSP is the set of domains of each variable, and is a singleton set:

$$\begin{aligned} D &= \{D_0, D_1, \ldots D_{63}\} \\ &= \{\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}\} \end{aligned} \tag{3.1}$$

3.2.3 Modeling Criterion **S-1**

"Each S-box has six bits of input and four bits of output."

Criterion **S-1** is implicit in the choice of variables. This is not discussed any further.

3.2.4 Modeling Criterion **S-3**

"If we fix the leftmost and rightmost input bits of the S-box and vary the four middle bits, each possible 4-bit output is attained exactly once as the middle four input bits range over their 16 possibilities."

Fixing the leftmost and rightmost input bits y_0y_5 to any of the possible four combinations, selects one of four subsets of the variables. Generation of constraints for clause **S-3** is now straightforward. All we require is that no two output variables, in each subset, should be equal. The inequalities are directly expressible as `Alldiff` constraints [49], [27]:

$$\begin{aligned} &\texttt{Alldiff}(x_0, x_2, x_4, ..., x_{30}) \\ &\texttt{Alldiff}(x_1, x_3, x_5, ..., x_{31}) \\ &\texttt{Alldiff}(x_{32}, x_{34}, x_{36}, ..., x_{62}) \\ &\texttt{Alldiff}(x_{33}, x_{35}, x_{37}, ..., x_{63}) \end{aligned}$$

For 6×4 S-boxes, each `Alldiff` constraint is expressible as $\frac{16 \times 15}{2} = 120$ binary constraints and all of these `Alldiff` constraints replace 480 binary inequality constraints. In general, for an $n \times m$ S-box organized as a $2^{n-m} \times 2^m$ table, there are 2^{n-m} `Alldiff` constraints for each row of the table. Each of these contains 2^m variables and is expressible as $\frac{2^m \times (2^m - 1)}{2}$ binary constraints. The total number of binary constraints that these `Alldiff` constraints replace

$$= 2^{n-m} \times \frac{2^m \times (2^m - 1)}{2} = 2^{n-1}(2^m - 1).$$

3.2.5 Modeling Criterion **S-4**

"If two inputs to an S-box differ in exactly one bit, the outputs must differ in at least two bits."

Consider any two 6-bit inputs i and j and their corresponding outputs $x_i, x_j \in D$, of a DES S-box S. Criterion **S-4** can be written in First-Order Logic as:

$$\begin{aligned}(\forall i)(\forall j)(0 \leq i < j \leq 63) \wedge wt(i \oplus j) = 1 \\ \Rightarrow wt(x_i \oplus x_j) \geq 2\end{aligned} \tag{3.2}$$

For a 6×4 S-box, each variable will participate in exactly 6 such binary constraints (one for each input bit), generating 192 binary constraints. For an $n \times m$ S-box, each variable participates in exactly n constraints. Since there are 2^n variables, the total number of constraints is equal to $2^n \times n$ and half of these constraints repeat due to symmetry. Therefore the number of binary constraints for criterion **S-4** is equal to $n \times 2^{n-1}$.

3.2.6 Modeling Criterion **S-5**

"If two inputs to an S-box differ in the two middle bits exactly, the outputs must differ in at least two bits."

Consider any two 6-bit inputs i and j and their corresponding outputs $x_i, x_j \in D$, of a DES S-box S. The fact that the 6-bit inputs i and j differ in the two middle bits implies that this 6-bit difference is exactly equal to 001100_2. **S-5** can be written in First-Order Logic as:

$$\begin{aligned}(\forall i)(\forall j)(0 \leq i, j \leq 63) \wedge (i \neq j) \wedge (i \oplus j = 001100_2) \\ \Rightarrow wt(x_i \oplus x_j) \geq 2\end{aligned} \tag{3.3}$$

For DES, this results in 32 binary constraints, each variable (S-box entry) participating in exactly one such binary constraint. For an $n \times m$ S-box, two n-bit inputs differ in their middle two bits in exactly one way. As such, each variable gives rise to exactly one binary constraint. For the 2^n variables there are 2^n constraints, and half of these repeat due to symmetry. Therefore, the number of binary constraints for criterion **S-5** is equal to 2^{n-1}. Note that n should be an even number, and $n \geq 2$.

3.2.7 Modeling Criterion **S-6**

“If two inputs to an S-box differ in their first two bits and are identical in their last two bits, the two outputs must not be the same."

Consider any two 6-bit inputs i and j and their corresponding outputs $x_i, x_j \in D$, of a DES S-box S. The fact that the 6-bit inputs i and j differ in their first two bits and are identical in their last two bits, implies that the input-difference $(i \oplus j) \wedge 110011_2$ is exactly equal to 110000_2. **S-6** can be written in First-Order Logic as:

$$\begin{aligned}(\forall i)(\forall j)(0 \leq i < j \leq 63), [(i \oplus j) \wedge 110011_2] = 110000_2 \\ \Rightarrow x_i \neq x_j\end{aligned} \qquad (3.4)$$

For DES, each variable is involved in 4 such binary constraints (one for each possible combination of the two middle input bits), resulting in a total of 128 new binary constraints.

For an $n \times m$ S-box, let us consider two n-bit inputs. Their first two bits differ and simultaneously, their last two bits are identical in exactly one way. The remaining $(n-4)$ bits can be selected in 2^{n-4} ways (these bits may or may not differ, nothing is said about them). Therefore each variable participates in 2^{n-4} binary constraints. For the 2^n variables, we have a total of $2^n \times 2^{n-4} = 2^{2n-4}$ constraints, half of which are identical due to symmetry. Therefore, for an $n \times m$ S-box, the number of binary constraints for criterion **S-6** is equal to $\frac{2^{2n-4}}{2} = 2^{2n-5}$. Note that $n \geq 4$.

3.2.8 Total Number of Binary Constraints

The total number of binary constraints, obtained by adding the above four results, is equal to $2^{n-1} \times (2^m + n + 2^{n-4})$, $n \geq 4$. For DES, this works out to 832 constraints. 160 of these binary constraints contain two variables that participated in other binary constraints in the set. After corresponding constraints are composed (refer Example 2.2), the total number of binary constraints formulated for criteria **S-3**, **S-4**, **S-5** and **S-6** is reduced to 672.

3.3 Summary and Looking Ahead

In this chapter we have formulated binary constraints for criteria **S-3**, **S-4**, **S-5** and **S-6** specified for DES S-box design. Criterion **S-1** is implicit in the choice of variables. In our experiments, these binary constraints have been precompiled into a solver (Section 2.7) that stores the binary constraints in extensional form (Section 2.6.1).

In the binary constraints modeled, their applicablity is a function of the S-box input size n for an $n \times m$ S-box. While all of them are applicable for 6×4 S-boxes such as those of DES, **S-5** does not make sense for 5×3 S-boxes $n = 5$ is not even. In the experiments discussed in Chapter 7, such criteria would have to be either modified or relaxed when running the solver on small-sized problems.

Criteria **S-2** and **S-7** are n-ary constraints. The key challenge is how to run the binary-constraint solver on these constraints *efficiently*. There are essentially two approaches for ensuring efficiency, both employing projections:

1. Project an n-ary constraint on past assignments and check if the n-ary constraint is satisfied *partially* for each assignment.

2. Project domain-values of future variables onto past assignments to completely eliminate checks for constraint satisfaction.

The second approach is better than the first, since, no explicit checking is required and all elements available after propagation are part of the solution space. Both heuristics are *incremental*, not checking on a complete S-box but on a *partial assignments*. Chapter 4 introduces the idea of a *partially assigned S-box*, and discusses the first approach applied to criteria **S-2** and **S-7**. Chapter 5 discusses the second approach applied to criterion **S-7**.

Criterion **S-8** deals with multiple S-boxes and cannot be modeled in this existing CSP framework. We discuss this criterion in Chapter 6. We have determined a way by which this criterion can be modeled as a pure CSP, by increasing the variables (and hence the solution space). This will be treated as future work and discussed in some detail in Chapter 9.

Chapter 4
The Nonlinearity Constraint

Criterion **S-2** is referred to as the *nonlinearity requirement.* The essence of this requirement is that the output of an S-box should be highly nonlinear. A measure of nonlinearity is derived in this Chapter, directly from Matsui's work on Linear Cryptanalysis of DES [34].

In this Chapter, we model S-box nonlinearity as a constraint. As we shall see, the result is a soft CSP. We will then formulate heuristics for speedup of S-box search to satisfy this criterion in addition to criteria **S-3** to **S-6**.

First, the nonlinearity requirement is repeated here for convenience:

S-2: *"No output bit of an S-box should be too close to a linear function of the input bits. (That is, if we select any output bit position and any subset of the six input bit positions, the fraction of inputs for which this output bit equals the exclusive-OR of these input bits should not be close to 0 or 1, but rather should be near $\frac{1}{2}$)."* [16]

Before proceeding any further with the nonlinearity analysis, let us discuss the idea of a *partially assigned* S-box and derive some properties of this kind of S-box. The derived properties are not present in current literature on S-boxes, and is one of the main contributions of this Dissertation.

4.1 Partially Assigned S-boxes

In the literature, each S-box entry or all entries are formed one bit at a time and these are often referred to as partial S-boxes. Often these entries are populated one *row* at a time or one *column* at a time [38]. In all of these cases, the S-box entries are formed bit by bit. In contrast, in our formulation each S-box entry is a complete number. However not all entries are *assigned* simultaneously or in other words, we do not always deal with completely *filled* S-boxes. In CSP terminology, this amounts to variables not all of which are being assigned immediately. This gives rise to *incremental* evaluations, which will be formulated for checking **S-2** and **S-7**. Throughout the rest of this Dissertation, we will refer to a Partially Assigned S-box to mean an assignment to a subset of the variables in X of the S-box.

Definition 4.1 (Partially Assigned S-box) *An S-box is* partially assigned *if and only if not all of its entries are assigned.*

Let X be the set of variables of an S-box and $X' \subseteq X$, the set of variables of a partially assigned S-box. We will denote a partial assignment to the variables in X of an S-box as follows. The tuple

$$A \;=\; \langle (x_0, d_0), (x_1, d_1), \ldots, (x_{|X'|-1}, d_{|X'|-1}) \rangle \tag{4.1}$$

corresponds to the partial assignment $x_0 = d_0, x_1 = d_1, \ldots, x_{|X'|-1} = d_{|X'|-1}$. The unassigned entries are given an invalid value (-1 in our implementations). In the illustrations for partial S-boxes these unassigned entries are not shown. This notation will be used in Chapter 7 to quantify the search points encountered during S-box search.

Example 4.1 *For the complete S-box S_8 of DES given in Figure 1.1, several partially assigned S-boxes are possible. A partial assignment to variables* $x_0 = 13, x_1 = 1, x_2 = 2, x_3 = 15, \ldots, x_{20} = 3$ *results in the partial S-box of Figure 4.1.*

13	2	8	4	6	15	11	1	10	9	3
1	15	13	8	10	3	7	4	12	5	

Figure 4.1: A partially assigned S-box obtained by assigning values to variables $x_0, x_1, \ldots, x_{20}$. Note that this assignment follows a zig-zag pattern.

As per the notation governed by equation 4.1, this assignment is equivalent to the tuple

$$A \;=\; \langle (x_0, 13), (x_1, 1), (x_2, 2), \ldots, (x_{19}, 5), (x_{20}, 3) \rangle$$

Notice that the way entries are populated depends on the manner in which the variables are organized with reference to Figure 3.1. Another fact worth observing is that the partially assigned S-box formation is contingent upon the *order* in which the variables are assigned. Example 4.2 illustrates this aspect.

Example 4.2 *In Chapter 5, we will consider a straight line variable ordering heuristic. In this heuristic, variables with even subscripts*

$x_0, x_2, \ldots, x_{30}$ are first assigned to fill the first row, followed by those with odd subscripts $x_1, x_3, \ldots, x_{31}$ to fill the second, and so on. In this configuration if we consider the following variable assignments, namely, $(x_0, x_2, x_4, x_6, x_8, x_{10}, x_{12}, x_{14}, x_{16}, x_{18}, x_{20}, x_{22}, x_{24}, x_{26}, x_{28}, x_{30}) = (13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7)$ and $(x_1, x_3, x_5, x_7, x_9, x_{11}) = (1, 15, 13, 8, 10, 3)$ then the partial S-box of Figure 4.2 will result.

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3										

Figure 4.2: A partially assigned S-box resulting from variables assigned using a straight-line pattern (row-wise)

As per the notation governed by equation 4.1, this assignment is equivalent to the tuple

$$\begin{aligned} A &= \langle (x_0, 13), (x_2, 2), (x_4, 8), (x_6, 4), (x_8, 6), (x_{10}, 15), (x_{12}, 11), (x_{14}, 1), \\ &\quad (x_{16}, 10), (x_{18}, 9), (x_{20}, 3), (x_{22}, 14), (x_{22}, 5), (x_{24}, 0), (x_{26}, 12), (x_{28}, 7), \\ &\quad (x_1, 1), (x_3, 15), (x_5, 13), (x_7, 8), (x_9, 10), (x_{11}, 3) \rangle \end{aligned}$$

In Chapter 3, we have denoted X as the set of variables for the S-box entries. For a partial S-box, we denote X' to be the variable-set, with $X' \subseteq X$. We will also denote a fully-filled S-box (which we will refer to simply as an S-box as has customarily been done) by Φ and a partially assigned S-box by Φ'.

4.2 *Analysis of Criterion* S-2

The rationale behind criterion **S-2** is to ensure that an S-box is highly non-linear. Matsui's work on linear cryptanalysis [34] uses a table called the Linear Approximation Table (LAT) that records the counts of linear combinations of all subsets of input and output bits, for a particular S-box. Consider an $n \times m$ S-box, i.e. one that for any n-bit input $i = i_0 \ldots i_{n-1}$ yields the m-bit output $x_i = x_{i_0} \ldots x_{i_{m-1}}$. The linear combinations to be checked for equality are obtained by selecting bits in i and x_i using selectors a and b respectively, where $0 \leq a < 2^n$ and $0 \leq b < 2^m$.

Example 4.3 *Let us consider a 6×4 S-box that takes as input $i = 46$ and gives an output $x_i = x_{46} = 13$. To check how close this input/output relationship is* *to a linear relationship, we need to fit a linear equation between the bits of* $i = 101110_2 = i_o i_1 i_2 i_3 i_4 i_5$ *and of* $x_i = 1101_4 = x_{i_0} x_{i_1} x_{i_2} x_{i_3}$*. If the bits 0, 2, 4 and 5 of input* i *and bits 1, 2 of output* x_i*, are selected, then the following equation is checked:*

$$\begin{aligned} i_0 \oplus i_2 \oplus i_4 \oplus i_5 &\overset{?}{=} x_{i_1} \oplus x_{i_2} \\ 1 \cdot i_0 \oplus 0 \cdot i_1 \oplus 1 \cdot i_2 \oplus 0 \cdot i_3 \oplus 1 \cdot i_4 \oplus 1 \oplus i_5 &\overset{?}{=} 0 \cdot x_{i_0} \oplus 1 \cdot x_{i_1} \oplus 1 \cdot x_{i_2} \oplus 0 \cdot x_{i_3} \\ a_0 i_0 \oplus a_1 i_1 \oplus a_2 i_2 \oplus a_3 i_3 \oplus a_4 i_4 \oplus a_5 i_5 &\overset{?}{=} b_0 x_{i_0} \oplus b_1 x_{i_1} \oplus b_2 x_{i_2} \oplus b_3 x_{i_3} \\ i.e.\ L_a(i) &\overset{?}{=} L_b(x_i) \\ or,\ L_{43}(46) &\overset{?}{=} L_6(13) \end{aligned}$$

where $a = a_0 a_1 a_2 a_3 a_4 a_5 = 101011_2 = 43$ *and* $b = b_0 b_1 b_2 b_3 = 0110_2 = 6$ *are selectors for the bits 0,2,4,5 of the S-box input* i *and bits 1,2 of S-box output* x_i*, respectively. This is a linear equation and if satisfied, the relationship as governed by this equation is indeed linear.*

We are not interested in merely one particular linear relationship. We want to check search for linear relationships in *all* possible selections of input and output bits of the S-box. The way of doing this is to enumerate all possible combinations (selections) among the $2^6 = 64$ selections of subsets of input bits in i versus the $2^4 = 16$ subsets of output bits of x_i and see how many of them are related by linear equations.

Example 4.4 *Consider the entry* $x_{46} = 13$ *of the* 6×4 *S-box of Example 4.3. We would like to know what the linear relationships between input 46 (*$= 101110_2$*)and output 13 (*$= 1101_2$*) exist. To do so, run the following check for* <u>*all*</u> *selectors* $a = a_0 a_1 a_2 a_3 a_4 a_5$, $b = b_0 b_1 b_2 b_3$, $1 \le a < 64, 1 \le b < 16$*:*

$$\begin{aligned} a_0 \cdot 1 \oplus a_1 \cdot 0 \oplus a_2 \cdot 1 \oplus a_3 \cdot 1 \oplus a_4 \cdot 1 \oplus a_5 \cdot 0 &\overset{?}{=} b_0 \cdot 1 \oplus b_1 \cdot 1 \oplus b_2 \cdot 0 \oplus b_3 \cdot 1 \\ i.e.\ L_a(i) &\overset{?}{=} L_b(x_i) \\ or,\ L_a(46) &\overset{?}{=} L_b(13) \end{aligned}$$

Note that we have excluded the selectors $a = 0, b = 0$ because if nothing is selected, the trivial and uninteresting result $0 = 0$ is encountered.

b	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a																
0	64	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32
1	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32
2	32	34	32	30	30	32	34	32	30	32	30	36	28	30	32	38
3	32	30	32	34	30	36	26	36	30	28	38	32	28	34	32	34
4	32	30	30	32	32	34	30	28	34	32	36	30	38	32	32	46
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
59	32	38	30	28	34	32	36	34	24	26	30	32	38	32	32	34
60	32	34	32	34	28	26	36	34	36	34	40	30	32	34	36	30
61	32	30	44	26	40	34	28	38	36	30	28	30	36	34	28	34
62	32	24	36	32	30	34	30	38	42	38	34	34	32	32	28	32
63	32	24	32	36	34	30	22	30	26	38	30	38	28	36	28	32

Table 4.1: Tabulating the counts $N_X^{\Phi}(a,b)$ for the S-box S_8 of Figure 1.1

4.2.1 Counting Linear Relationships in a Completely Filled S-box: $N_X^{\Phi}(a,b)$

For a given S-box Φ with all variables in X, let us define $N_X^{\Phi}(a,b)$ as follows:

$$N_X^{\Phi}(a,b) = \#\{i : L_a(i){=}L_b(x_i); a, i \in \mathbb{Z}_{2^n}; b, x_i \in \mathbb{Z}_{2^m}\} \tag{4.2}$$

where $L_\omega(x)$ is defined in Equation 2.10. Equation 4.2 suggests that linear combinations of input and output bits of all entries in an S-box are *counted*. The minimum value of $N_X^{\Phi}(a,b)$ is zero and the maximum value is 2^n. The values of $N_X^{\Phi}(a,b)$ are tabulated in a $2^n \times 2^m$ matrix. Example 4.5 illustrates the tabulation process.

Example 4.5 *Consider DES S-box S_8 of Figure 1.1. This is a 6×4 S-box with $n = 6$ and $m = 4$. Form a table the rows of which are indexed by a and the columns, by b, where $a \in \mathbb{Z}_{64}$, and $b \in \mathbb{Z}_{16}$. Table 4.1 displays the table having $2^6 = 64$ rows and $2^4 = 16$ columns having a total of $64 \times 16 = 1024$ entries. Only the first 4 and last 4 rows of the table are shown in the figure, along with all the columns.*

Let us study the entry corresponding to Row 3, Column 5 of the table, for which $a = 3 = 000011_2 = a_0a_1a_2a_3a_4a_5$ and $b = 5 = 0101_2 = b_0b_1b_2b_3$. This entry is equal to the total number of times the following equation is satisfied for all 64 4-bit entries $x_i = x_{i0}x_{i1}x_{i2}x_{i3}$ of S-box S_8 corresponding to each 6-bit input $i = i_0i_1i_2i_3i_4i_5$, $0 \le i < 64$:

$$
\begin{aligned}
L_3(i) &= L_5(x_i) \\
0 \cdot i_0 \oplus 0 \cdot i_1 \oplus 0 \cdot i_2 \oplus 0 \cdot i_3 \oplus 1 \cdot i_4 \oplus 1 \cdot i_5 &= 0 \cdot x_{i0} \oplus 1 \cdot x_{i1} \oplus 0 \cdot x_{i2} \oplus 1 \cdot x_{i3} \\
i_4 \oplus i_5 &= x_{i1} \oplus x_{i3} \qquad (4.3)
\end{aligned}
$$

The entry in the table under row 3, column 5 reads 36, which means that for 36 entries out of the 64 in DES S-box S_8, Equation 4.3 is satisfied. The maximum value of an entry is 64 since S_8 contains 64 entries. This process is repeated for all entries (a, b) in the table to yield a count for every possible linear combinations of the bits in an S-box input and its corresponding output.

4.2.2 Counting Linear Relationships in a Partially Assigned S-box: $N_{X'}^{\Phi'}(a,b)$

In Matsui's work, the quantity $N_X^{\Phi}(a,b)$ is specified for an S-box that has all its entries filled. Given a partial $n \times m$ S-box Φ' and variable-set $X' \subseteq X$, let us define another quantity $N_{X'}^{\Phi'}(a,b)$ as follows:

$$N_{X'}^{\Phi'}(a,b) = \#\{i : L_a(i){=}L_b(x_i); x_i \in X'; a \in \mathbb{Z}_{2^n}; b, x_i \in \mathbb{Z}_{2^m}\}$$

4.2.3 Properties of $N_{X'}^{\Phi'}(a,b)$

Besides Matsui's properties for $N_{X'}^{\Phi'}(a,b)$ [34], the following properties also follow from the definition of $N_{X'}^{\Phi'}(a,b)$.

Property 4.1 *For any a,b,X',Φ', $0 \leq N_{X'}^{\Phi'}(a,b) \leq |X'|$.*

This property follows at once from the fact that $N_{X'}^{\Phi'}(a,b)$ is a count, which ranges between the minimum value of 0 and the maximum value of $|X'|$.

Property 4.2 *For any a,b,u,X',Φ', $N_{X' \cup \{u\}}^{\Phi'}(a,b) - N_{X'}^{\Phi'}(a,b) \in \{0,1\}$.*

This property follows at once since, adding an S-box entry due to the assignment to a variable $u \notin X'$ results in the counts of equalities of linear combinations of S-box inputs and outputs either increasing by 0 or 1.

Let us examine, through an example, the manner in which $N_X'^{\Phi'}(a,b)$ progresses as each variable of an S-box is assigned a value. We will also be able to understand the properties of $N_{X'}^{\Phi'}(a,b)$ just listed. We assume that the S-box entries are assigned in the manner in Example 4.1.

b	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a																
0	1	0	1	0	0	1	0	1	0	1	0	1	1	0	1	0
1	1	0	1	0	0	1	0	1	0	1	0	1	1	0	1	0
2	1	0	1	0	0	1	0	1	0	1	0	1	1	0	1	0
3	1	0	1	0	0	1	0	1	0	1	0	1	1	0	1	0
4	1	0	1	0	0	1	0	1	0	1	0	1	1	0	1	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
59	1	0	1	0	0	1	0	1	0	1	0	1	1	0	1	0
60	1	0	1	0	0	1	0	1	0	1	0	1	1	0	1	0
61	1	0	1	0	0	1	0	1	0	1	0	1	1	0	1	0
62	1	0	1	0	0	1	0	1	0	1	0	1	1	0	1	0
63	1	0	1	0	0	1	0	1	0	1	0	1	1	0	1	0

Table 4.2: Tabulating the counts $N_{X'}^{\Phi'}(a,b)$ for a partially assigned S-box Φ' with only the first entry $x_0 = 13$ is populated.

Example 4.6 *Consider again, the DES* 6×4 *S-box* S_8 *of Figure 1.1. Assign only the variable* $x_0 = 13$ *with the first entry in the S-box, with all remaining variables unassigned. For this partially assigned S-box, let us organize a table exactly in the manner presented in Example 4.5, resulting in Table 4.2.*

As one can observe, assigning a value to a *single* variable results in a table for $N_{X'}^{\Phi'}(a,b)$ that contains entries having either 0 or 1. That is because for a single assignment, either a linear combination of subsets of input bits versus output bits results in either an equality (entry 1) or an inequality (entry 0). With one more variable assigned (that is, two variables assigned), the resulting table is obtained by adding the (0-1) tables of the individual entries. Another way of stating this is that the entries in the table due to the previous assignment is increased by at most 1 to form the cumulative table. This is what Property 4.2 states.

The new cumulative table formed will now have entries that range between 0 and 2. In general, when a partially assigned S-box contains $|X'|$ entries, its $N_{X'}^{\Phi'}(a,b)$ entry ranges from 0 up to $|X'|$. This is what Property 4.1 states.

Example 4.7 *The values for* $N_{X'}^{\Phi'}(a,b)$ *of the partially assigned S-box* Φ' *of Figure 4.1, with assignments made to variables* $X' = \{x_0, x_1, \ldots, x_{20}\}$*, is displayed in Table 4.3.*

Observe that the entries range between 0 and 21. In fact, the minimum entry is 5 while the maximum entry is 21. Also observe that an entry may be even or

b	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a																
0	21	9	11	11	11	11	9	9	10	10	10	10	10	6	10	10
1	11	11	9	9	13	9	11	11	10	14	10	10	10	10	10	10
2	11	11	9	9	11	11	9	13	8	8	8	12	10	10	18	10
3	11	11	9	13	11	11	9	17	10	10	10	10	8	8	8	12
4	12	10	8	6	10	12	10	8	9	11	11	9	13	11	11	17
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
59	10	10	12	12	10	14	16	8	5	9	9	9	11	11	11	11
60	9	7	9	11	11	9	11	9	14	12	12	10	14	12	12	6
61	11	9	11	9	13	11	5	15	10	8	8	10	14	12	12	10
62	11	5	11	13	11	9	11	13	12	10	10	12	10	12	8	10
63	11	9	11	13	7	9	7	9	6	16	12	14	12	10	10	12

Table 4.3: Tabulating the counts $N_{X'}^{\Phi'}(a, b)$ for the partially assigned S-box Φ' of Figure 4.1.

odd. In contrast, for an S-box having all entries, the $N_X^{\Phi}(a, b)$ entries are always even.

4.2.4 A Measure of Nonlinearity of an S-box

For selectors a and b defined as above, let $p(a, b)$ denote the fraction of cases when $L_a(i) = L_b(x_i)$, computed as:

$$p(a, b) = \frac{N_X^{\Phi}(a, b)}{2^n} \tag{4.4}$$

If $p(a, b)$ is equal to 1, this indicates that the linear combination of the output bits selected by b equals a linear combination of the input bits selected by a, i.e., $\forall i, L_a(i) = L_b(x_i)$.

If $p(a, b)$ is equal to zero, then the linear combination of the output bits selected by b is never equal to the linear combination of input bits selected by a. DES criterion **S-2** (and the stronger criterion **S-2'**) stipulates that $p(a, b)$ should be near $\frac{1}{2}$ for all a, b.

We are interested in how close $p(a, b)$ is to $\frac{1}{2}$. Let $\rho(a, b)$ denote the absolute value of the difference between these quantities. Then,

$$\rho(a, b) = |p(a, b) - \frac{1}{2}| \tag{4.5}$$

Multiplying equation 4.5 by 2^n, we have the following:

$$2^n \times \rho(a,b) = 2^n \times |p(a,b) - \frac{1}{2}| = |N_X(a,b) - 2^{n-1}|$$

by substituting for $N_X(a,b)$ from equation 4.4. DES criterion **S-2** (and **S-2**') alternatively stipulates that $|N_X(a,b) - 2^{n-1}|$ for all selector-pairs (a,b) should be as close to zero as possible.

4.2.5 The Score of an S-box

The ideal case where $|N_X^\Phi(a,b) - 2^{n-1}|$ is exactly equal to zero for all selector-pairs (a,b), has so far not been attained in the literature for common cryptosystems. The most effective linear approximation of a DES S-box is obtained if, for some a and b, $|N_X^\Phi(a,b) - 2^{n-1}|$ is maximal. To reduce the weakest point of the S-box, we use the so called *effectiveness* of linearization [46] of an S-box Φ as the score, $\sigma_X(\Phi)$, given by the maximum value of $|N_X(a,b) - 2^{n-1}|$ over all (a,b):

$$\sigma_X(\Phi) = max\{|N_X^\Phi(a,b) - 2^{n-1}| : 1 \leq a < |X|; 1 \leq b < |D|\} \tag{4.6}$$

It can be easily observed that an S-box with a smaller score is considered better (i.e. less linear and more nonlinear).

Matsui [34] considered the general case when b is not a power of 2, corresponding to the criterion **S-2'** that is stricter than **S-2**. Coppersmith [16] labelled **S-2'** as an *additional* property not originally used in the design of the S-boxes for DES, and we will adhere to the same premise accordingly, in our work. *Therefore, in Equation 4.6, we will always assume that b is a power of 2.*

4.2.6 The Linear Approximation Table (LAT)

The linear approximation table [34] for an $n \times m$ S-box is a $2^n \times 2^m$ matrix [34]. Its rows are headed by selector a $(0 \leq a < 2^n)$ and columns, by selector b $(0 \leq b < 2^m)$. Each entry is equal to the value of $N_X(a,b) - 2^{n-1}$ (including its sign). The entry, in row a and column b represent a measure of the correlation between the input bits selected by a and the output bits selected by b. Properties of the table, and of $N_X^\Phi(a,b)$, are discussed in [34, 46, 26].

As Equation 4.6 suggests, the score $\sigma_X(\Phi)$ of the $n \times m$ S-box Φ is obtained by taking the maximum value of the absolute values of all entries falling under those columns whose heading is a power of two.

b	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a																
0	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	2	0	-2	-2	0	2	0	-2	0	-2	4	-4	-2	0	6
3	0	-2	0	2	-2	4	-6	4	-2	-4	6	0	-4	2	0	2
4	0	-2	-2	0	0	2	-2	-4	2	0	4	-2	6	0	0	14
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
59	0	6	-2	-4	2	0	4	2	-8	-6	-2	0	6	0	0	2
60	0	2	0	2	-4	-6	4	2	4	2	8	-2	0	2	4	-2
61	0	-2	12	-6	8	2	-4	6	4	-2	-4	-2	4	2	-4	2
62	0	-8	4	0	-2	2	-2	6	10	6	2	2	0	0	-4	0
63	0	-8	0	4	2	-2	-10	-2	-6	6	-2	6	-4	4	-4	0

Table 4.4: The Linear Approximation Table for DES S-box S_8 of Figure 1.1

Example 4.8 *The first five and last five rows of the Linear Approximation Table for the DES* 6×4 *S-box* S_8*, with* $n = 6$ *and* $m = 4$*, are shown in in Table 4.4. Each entry in this table is equal to* $N_X(a,b) - 2^{n-1} = N_X(a,b) - 2^5 = N_X(a,b) - 32$*. In other words, the table constructed in this example follows straightforwardly from that of Example 4.5 (Table 4.1) by subtracting 32 from each entry of the latter table.*

The score of DES S-box **S-8** *is obtained by taking the maximum values of the absolute values of the entries under columns 1, 2, 4 and 8 that are powers of two, which is equal to 12.*

4.2.7 The Score of a Partially Assigned S-box

The score $\sigma_{X'}, X' \subseteq X$, of a partially assigned $n \times m$ S-box Φ' is defined as follows:

$$\sigma_{X'}(\Phi') = max\{|N_{X'}^{\Phi'}(a,b) - 2^{n-1}| : 1 \leq a < 2^n; 1 \leq b < |2^m|\}$$

4.3 Modeling Criterion **S-2***: A Non-incremental, Complete Heuristic*

The criteria **S-2** leads to a soft constraint that minimizes $\sigma_X(\Phi)$. When implemented as a hard constraint for a threshold τ, it has the form:

$$\sigma_X(\Phi) \leq \tau \tag{4.7}$$

The functions the procedure `Solver` of Algorithm 1 are listed in Table 4.5. Functions `InitialPartialVarsAndCounts()` and `CheckPartialSBox()` are used in incremental heuristics and therefore, there is nothing to be done for this heuristic. Function `ProcessNary()` only checks if all variables are instantiated, returning **true** if so, and **false** otherwise. This function, along with functions `ReverseUpdateCountDistSet()` and `ReverseUpdateCount()`, is designed to check for criterion **S-7**, discussed in detail in Chapter 5.

Function `MakeAndCheckSBox()` calls function `MakeSBox()` that prepares an S-box Φ from the assignment to all variables in X. Next, `MakeAndCheckSBox()` verifies Equation refEq:s2 and if this equation is satisfied, verifies if the constraints for criterion **S-7** are also satisfied (refer Chapter 5). If both are satisfied, this function returns the S-box Φ that satisfies all modeled constraints, which is printed in `Solver` (Algorithm 1, Line 9).

Function	**Description**
`InitPartialVarsAndCounts()`	(No operation)
`CheckPartialSBox()`	return **true**
`ProcessNary()`	return ($k = \lvert X \rvert$).
`MakeAndCheckSBox()`	$\Phi \leftarrow$ `MakeSBox`(X) Compute $\sigma_X(\Phi)$ using Equation 4.6 if $(\sigma_X(\Phi) \leq \tau) \wedge$ (**S-7** is satisfied) (refer Chapter 5) return Φ
`ProcessOtherDomains()`	return **true**
`ReverseUpdateCountDistSet()`	(No operation)
`ReverseUpdateCount()`	(No operation)

Table 4.5: Functions for the Nonincremental, Complete Heuristic for **S-2**, in `Solver`

The constraint of Equation 4.7 is not implemented using an extensional representation. Rather, a specialized function is added to the solver that works with a 2^{n+m} size storage, replicated at each level in the search tree. This results in a total space requirement of 2^{2n+m} bytes. For DES boxes the constraint requires $64kB$.

4.4 *Modeling Criterion* **S-2***: An Incremental, Incomplete Check*

Equation 4.7 suggests a non-incremental approach to **S-2** checking. Experiments discussed in Chapter 7 reveal a very inefficient S-box search using this heuristic as expected.

We formulate an incremental check by which after *each* variable is assigned, determine $\sigma_{X'}$ and repeat so long as Equation 4.7 is not violated. As soon as this equation is violated, we backtrack. This approach has significantly speeded up the search for S-boxes. Moreover, it led to 6×4 S-boxes that yielded values of σ_X superior to those for the published eight DES S-boxes. Table 4.6 implements the functions for this heuristic in pseudocode form.

Function	**Description**
`InitPartialVarsAndCounts`()	Let $X' \leftarrow \{x : x \in X \wedge x \text{ is assigned.}\}$
`CheckPartialSBox`()	$\Phi' \leftarrow$ `MakePartialSBox`(X') Compute $\sigma'_X(\Phi')$ using Equation 4.7 return $\leftarrow (\sigma_{X'}(\Phi') \leq \tau)$
`ProcessNary`()	return $(k = \|X\|)$
`MakeAndCheckSBox`()	$\Phi \leftarrow$ `MakeSBox`(X)
`ProcessOtherDomains`()	(No operation)
`ReverseUpdateCountDistSet`()	(No operation)
`ReverseUpdateCount`()	(No operation)

Table 4.6: Functions for the Incremental, Incomplete Heuristic for **S-2**, in `Solver`

The function `InitPartialVarsAndCounts()` keeps track of the variables that are assigned, in X'. The function `CheckPartialSBox()` actually performs the verification of Equation 4.7. However, the check for violation is done in `Solver` (Algorithm 1), in the if-statement after Step 6. Function `ProcessNary()` simply checks if all variables have been instantiated. It is actually designed to perform checks for the global n-ary constraint for **S-7**, and is discussed in Chapter 5. So are functions `ProcessOtherDomains()`, `ReverseUpdateCountDistSet()` and `ReverseUpdateCount()`. In the end, by the time Line 9 is encountered, all variables are assigned and criterion **S-2** is satisfied (not necessarily **S-7**), and the S-box is output.

Despite its efficacy, this heuristic is *incomplete*. The reason is that although $\sigma_{X'}(\Phi')$ exceeded the threshold τ on some partial assignment to variables in X', *it is not necessary that upon the next assignment, $\sigma_{X'}(\Phi')$ will monotonically increase* *to always exceed* τ. (Property 4.2 suggests that $N_{X'}^{\Phi'}(a,b)$ increases monotonically – but not strictly – by at most 1.) In fact, $\sigma_{X'}(\Phi')$ is often found to *decrease* during subsequent assignments, which this heuristic does not catch. This partial S-box should not always be abandoned.

We further formulate a *complete* heuristic by providing a characterization for a partial S-box Φ' to extend to a complete S-box Φ. The trick to obtaining the condition is to not study the progress of $\sigma_{X'}(\Phi')$, but to analyze so for $N_{X'}^{\Phi'}(a,b)$.

4.5 Modeling Criterion S-2: An Incremental, Complete Check using Soft Constraint Decomposition

We will now project the soft constraint of Eq. 4.7 onto hard constraints involving ϕ variables, ϕ being the number of variables instantiated to form a partially assigned S-box Φ' and $\phi \leq |X|$. During projection, the goal is for the final score of S-box Φ to not exceed the maximum threshold τ:

$$\max_{a,b} |N_X^{\Phi}(a,b) - \frac{|X|}{2}| \leq \tau \tag{4.8}$$

4.5.1 Construction of a Partially assigned S-box

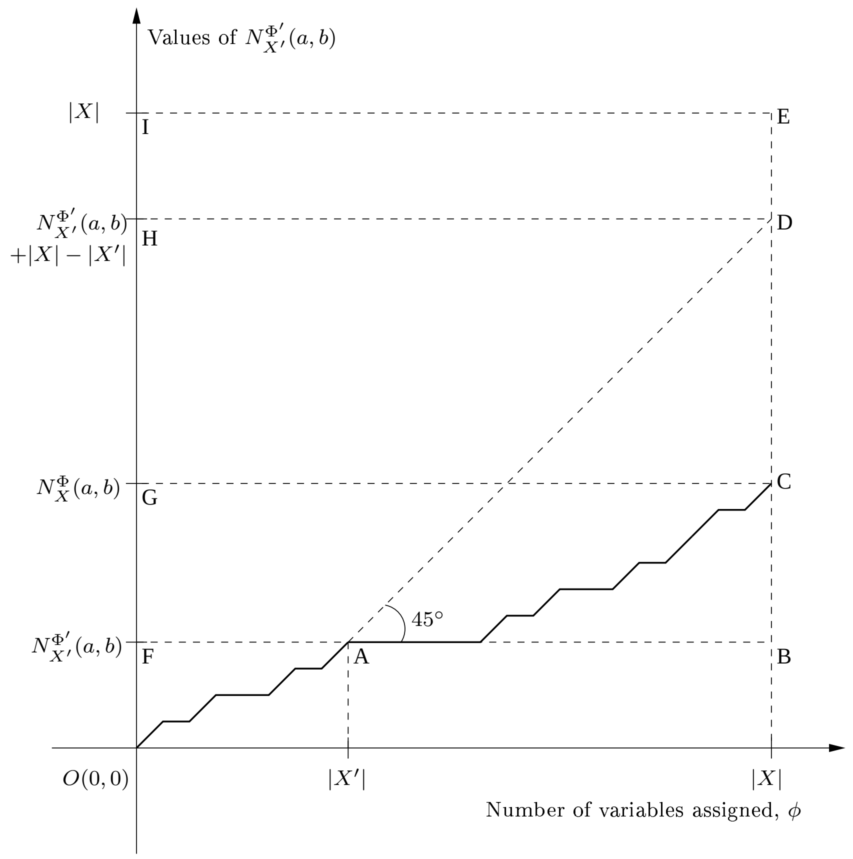
Figure 4.3 depicts the distribution of the counts $N_{X'}^{\Phi'}(a,b)$ on one selector-pair (a,b), for a partially instantiated S-box Φ'. The horizontal axis is the number of variables instantiated, ϕ. After $|X'|$ variables are instantiated at point A along the solid line, the dashed line at a 45-degree angle with the horizontal represents the pathological case where the score $\sigma_{X'} = N_{X'}^{\Phi'}(a,b)$ increases by one for every subsequent extension of Φ' up to point D. The solid zig-zag lines connecting points A and C represents the corresponding, "actual" distribution of $N_{X'}^{\Phi'}(a,b)$ for the complete S-box Φ to attain the score equal to $\sigma_{X'} = N_X^{\Phi}(a,b)$ at point C. From this construction, we have $OF = N_{X'}^{\Phi'}(a,b)$, $OG = N_X^{\Phi}(a,b)$, $FH = BD = AB = |X| - |X'|$, and $OH = OF + FH = N_{X'}^{\Phi'}(a,b) + |X| - |X'|$.

4.5.2 Lower and Upper bounds for $\max_{a,b} N_{X'}^{\Phi'}(a,b)$

Let us observe two properties of partially assigned S-boxes.

Property 4.3 *A partially instantiated S-box* Φ' *with values for variables in* $X', X' \subseteq X$*, cannot be extended to a solution with score better than a threshold* τ *if the following inequality is not satisfied:*

$$\max_{a,b} N_{X'}^{\Phi'}(a,b) \geq |X'| - \tau - \frac{|X|}{2} \tag{4.9}$$



Values of $N^{\Phi'}_{X'}(a,b)$
$|X|$
I
E
$N^{\Phi'}_{X'}(a,b)$ $+|X|-|X'|$
H
D
$N^{\Phi}_{X}(a,b)$
G
C
45°
$N^{\Phi'}_{X'}(a,b)$
F
A
B
$O(0,0)$
$|X'|$
$|X|$
Number of variables assigned, ϕ

Figure 4.3: Evaluating partially instantiated S-boxes.

Proof By construction, $(|X| - |X'|)$ remaining variables are to be instantiated in order to extend Φ' to Φ. To guarantee extensibility, the following inequality should hold:

$$\begin{aligned} OG &\leq OH \\ N^{\Phi}_{X}(a,b) &\leq N^{\Phi'}_{X'}(a,b) + |X| - |X'| \end{aligned}$$

This is true for all selectors a and b, and in particular, holds for the maximum value of $N^{\Phi}_{X}(a,b)$ (resp. $N^{\Phi'}_{X'}(a,b)$) over all a, b:

$$\max_{a,b} N^{\Phi}_{X}(a,b) \leq |X| - |X'| + \max_{a,b} N^{\Phi'}_{X'}(a,b) \tag{4.10}$$

From the goal specified by Equation 4.8,

$$\begin{aligned} \frac{|X|}{2} - \max_{a,b} N_X^{\Phi}(a,b) &\leq \tau \\ \text{i.e. } \frac{|X|}{2} - \tau &\leq \max_{a,b} N_X^{\Phi}(a,b) \end{aligned} \tag{4.11}$$

Combining Eq. 4.10 and Eq. 4.11,

$$\frac{|X|}{2} - \tau \leq \max_{a,b} N_X^{\Phi}(a,b) \leq |X| - |X'| + \max_{a,b} N_{X'}^{\Phi'}(a,b)$$

By transitivity and regrouping,

$$\begin{aligned} \max_{a,b} N_{X'}^{\Phi'}(a,b) &\geq \frac{|X|}{2} - \tau - |X| + |X'| \\ \text{i.e. } \max_{a,b} N_{X'}^{\Phi'}(a,b) &\geq |X'| - \tau - \frac{|X|}{2} \end{aligned}$$

Q.E.D.

Property 4.4 *A partially instantiated S-box Φ' with values for variables in $X', X' \subseteq X$, cannot be extended to a solution with score better than a threshold τ if the following inequality is not satisfied:*

$$\max_{a,b} N_{X'}^{\Phi'}(a,b) \leq \frac{|X|}{2} + \tau \tag{4.12}$$

Proof Given a partial S-box assignment Φ' with variables in X', by the end of the construction of any solution Φ obtained by extending Φ', the following inequality holds:

$$\begin{aligned} OF &\leq OG \\ \text{i.e. } N_{X'}^{\Phi'}(a,b) &\leq N_X^{\Phi}(a,b) \end{aligned} \tag{4.13}$$

This is true for all selectors a and b, and in particular, holds for the maximum value of $N_{X'}^{\Phi'}(a,b)$ (resp. $N_X^{\Phi}(a,b)$) over all a, b:

$$\max_{a,b} N_{X'}^{\Phi'}(a,b) \leq \max_{a,b} N_X^{\Phi}(a,b) \tag{4.14}$$

From the goal specified by Eq. 4.8,

$$\begin{aligned} \max_{a,b} N_X^{\Phi}(a,b) - \frac{|X|}{2} &\leq \tau \\ \text{i.e. } \max_{a,b} N_X^{\Phi}(a,b) &\leq \frac{|X|}{2} + \tau \end{aligned} \tag{4.15}$$

Combining Eq. 4.14 and 4.15,

$$\max_{a,b} N_{X'}^{\Phi'}(a,b) \leq \max_{a,b} N_X^{\Phi}(a,b) \leq \frac{|X|}{2} + \tau$$

The result follows by transitivity.

Q.E.D.

Eq. 4.9 and Eq. 4.12 facilitate decomposition of the soft constraint of Eq. 4.7 into hard constraints. Once a partial S-box Φ' is constructed with $\phi = |X'| < \frac{|X|}{2}$ variables assigned, $H_{C_7}^{\phi,\tau}$ checks to see if Φ' satisfies the above two inequalities. If not, Φ' is rejected otherwise Φ' is extended by instantiating the next variable and the checks are repeated. The process goes on until an S-box Φ with all variables assigned, is obtained by which time **S-2** is now already satisfied.

4.5.3 An Example

An example is provided to understand the working of this heuristic.

Example 4.9 *We find in Chapter 7 that the maximum score for all of the DES S-boxes of Figure 1.1 is equal to 18. Consider that our threshold* τ *for a* 6×4 *S-box* Φ *is 16. Since there are* $2^6 = 64$ *variables* $X = \{x_0, x_1, \ldots, x_{63}\}$, *the partial check should begin after* $\frac{|X|}{2} + \tau$ *variables are assigned, that is, after* $(32 + 16) = 48$ *variables are assigned. Let us examine the progress of* $\max_{a,b} N_{X'}^{\Phi'}(a,b)$ *and the score as we incrementally assign to the remaining 16 variables, starting from* x_{48}.

For each variable assigned, Table 4.9 populates the forms assumed by Eq. 4.9 and Eq. 4.12.

Thus, after x_{63} *is assigned to yield S-box* Φ *with all entries filled in,* $\max_{a,b} N_X^{\Phi}(a,b)$ *will range between 16 and 48. As a result, its score will always range between 0 and 16.*

4.5.4 The Solver that Implements this Heuristic

The incremental, complete checking heuristic is implemented in the solver `Solver` of Algorithm 1, for which the functions of the latter algorithm are described in Table 4.8.

Variable assigned	**Number of variables** $\lvert X'\rvert$	**Inequalities 4.9 and 4.12**
x_{48}	49	$1 \le \max_{a,b} N_{X'}^{\Phi'}(a,b) \le 48$
x_{49}	50	$2 \le \max_{a,b} N_{X'}^{\Phi'}(a,b) \le 48$
x_{50}	51	$3 \le \max_{a,b} N_{X'}^{\Phi'}(a,b) \le 48$
x_{51}	52	$4 \le \max_{a,b} N_{X'}^{\Phi'}(a,b) \le 48$
$\vdots$	$\vdots$	$\vdots$
x_{60}	61	$13 \le \max_{a,b} N_{X'}^{\Phi'}(a,b) \le 48$
x_{61}	62	$14 \le \max_{a,b} N_{X'}^{\Phi'}(a,b) \le 48$
x_{62}	63	$15 \le \max_{a,b} N_{X'}^{\Phi'}(a,b) \le 48$
x_{63}	64	$16 \le \max_{a,b} N_{X'}^{\Phi'}(a,b) \le 48$

Table 4.7: Progress made by the incremental checks after assignments to the first 48 variables, for a 6×4 S-box with a threshold score of 16 sought

The implementation of this heuristic is very similar to that of the incomplete, incremental check of Table 4.6, with the difference in function `CheckPartialSBox()`. In this heuristic, this function is *conditionally* called only after the first $(\frac{X}{2} + \tau)$ variables are assigned. In that case, the function `CheckPartialSBox()` returns the results of the verification of Eq. 4.9 and Eq. 4.12. However, the check for violation is done in `Solver` (subsection 2.7), in the if-statement after Step 6. Just as in the case for the incomplete heuristic, function `ProcessNary()` simply checks if all variables have been instantiated. It is actually designed to perform checks for the global n-ary constraint for **S-7**, discussed in Chapter 5. So are functions `ProcessOtherDomains()`, `ReverseUpdateCountDistSet()` and `ReverseUpdateCount()`. In the end, by the time Line 9 is encountered, all variables are assigned and criterion **S-2** is satisfied (not necessarily **S-7**), and the S-box is output.

This heuristic is complete and finds all solutions, now that the progress of $\max_{a,b} N_{X'}^{\Phi'}(a,b)$ is kept track of, instead of the score $\sigma'_X(\Phi')$ of the partially assigned S-box Φ'.

4.6 Summary of Heuristics and Looking Ahead

We have discussed the following heuristics for DES criterion **S-2**:

1. A non-incremental, complete heuristic, namely, the generate-and-test approach to satisfying criterion **S-2** by verifying Eq. 4.7.

Function	Description
`InitPartialVarsAndCounts()`	Let $X' \leftarrow \{x : x \in X \wedge x$ is assigned.$\}$

`CheckPartialSBox()`	**if** $(k \geq \frac{\lvert X\rvert}{2} + \tau)$ **then**
	  s2ok $\leftarrow$ **false**
	  $\Phi' \leftarrow$ `MakePartialSBox`(X')
	  Determine $M \leftarrow max_{a,b} N_{X'}^{\Phi'}(a, b)$
	  s2ok $\leftarrow (\lvert X'\rvert - \tau - \frac{\lvert X\rvert}{2} \leq M \leq \frac{\lvert X\rvert}{2} + \tau)$
	end if
	return `s2ok`

`ProcessNary()`	return $(k = \lvert X\rvert)$
`MakeAndCheckSBox()`	$\Phi \leftarrow$ `MakeSBox`(X)
`ProcessOtherDomains()`	return **true**
`ReverseUpdateCountDistSet()`	(No operation)
`ReverseUpdateCount()`	(No operation)

Table 4.8: Functions for the Incremental, Complete Heuristic for **S-2**, in `Solver`

2. An incremental, incomplete heuristic to partial S-box checking, namely, one in which **S-2** is checked by verifying Eq. 4.7 for every partial assignment. This heuristic abandons partial solutions that violate Eq. 4.7 that may have actually become solutions upon extension.

3. An incremental, complete heuristic to partial S-box checking, namely, one in which **S-2** is checked by verifying Eq. 4.9 and Eq. 4.12.

Experiments will reveal significant speedup of the incremental heuristic over the non-incremental approach (both complete). The incremental but incomplete approach will be seen to generate 6×4 S-boxes superior to the existing, published DES S-boxes. This is not observed in the case of either of the two complete heuristics developed in this Chapter. All of these results are presented in Chapter 7.

We next discuss development of heuristics for criterion **S-7** which we will call as the *COUNT* constraint. The functions implemented in this Chapter will be repeated in Chapter 5, but now the functions designed for **S-7** will also be in place, rendering full implementations of heuristics to generate solutions that satisfy all criteria.

Chapter 5

Decomposition of Global Constraints, and Heuristics

S-7: *"For any nonzero 6-bit difference between inputs $\Delta I_{i,j}$, no more than eight of the 32 pairs of inputs exhibiting $\Delta I_{i,j}$ may result in the same output difference $\Delta O_{i,j}$."* [16]

We now formulate criterion **S-7** as a constraint. This criterion deals with *counts* of differences (Hamming distances) between pairs of variables. As such, we will often refer to the resulting constraint as the *COUNT* constraint.

The COUNT constraint is a global n-ary constraint. It is n-ary because it involves participation of more than two variables. It is global because it can be decomposed into smaller-arity constraints (not necessarily binary). However, this decomposition is not straightforward because the variables themselves are not being split. A *function* on the variables, namely *counts* of the differences between pairs of variables is being considered for splitting.

In spite of this infeasibility of straightforward splitting, we demonstrate an elegant way by which projection is employed to achieve the domain-reductions resulting in S-box speedup.

This chapter outlines three heuristics for **S-7**: A non-incremental heuristic (generate-and-test), a simple incremental heuristic (taking advantage of the incremental heuristic for **S-2**), and the domain-reduction heuristic for **S-7** that employs projections.

An optimization, introduced in `Solver` (Section 2.7), is discussed in Section 5.5.1. The function `SelectNextVariable` in `Solver` selects the next variable depending upon the type of variable-ordering heuristic employed. This function is amplified in Section 5.5.2. Two heuristics are considered there, namely, a *Straight Line* variable-ordering heuristic (the default ordering of arranging from left to right in an S-box such as the one in Figure 3.1), and a *Zig-Zag* variable-ordering heuristic.

S_8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Figure 5.1: DES S-box S_8 used in Example 5.1

5.1 *Modeling criterion* S-7

We formulate **S-7** for an $n \times m$ S-box. Only input pairs $(i, 2^n - 1 - i)$, $0 \leq i < 2^{n-1}$, to the S-box differ by all n bits. Consider the set $O_7 = \{(x_i, x_{2^n-1-i}) : 0 \leq i < 2^{n-1})\}$ of pairs of outputs corresponding to these input-pairs, with $|O_7| = 2^{n-1}$. Criterion **S-7** applies to m-bit differences $d = x_i \oplus x_{2^n-1-i}$, $0 \leq d < 2^m$.

Let $f : \mathbb{Z}_{2^m} \to \mathbb{Z}_{2^{n-1}}$ denote a *count* function, with $f(d)$ signifying the *frequency of occurrence* of an m-bit number $d = x_i \oplus x_{2^n-1-i}$ where $(x_i, x_{2^n-1-i}) \in O_7$, $0 \leq i < 2^{n-1}$. Note that

$$\Sigma_{i=0}^{2^{n-1}-1} f(x_i \oplus x_{2^n-1-i}) = 2^{n-1}.$$

According to **S-7**, no more than eight elements in O_7 should evaluate to the same m-bit difference d. Criterion **S-7** is formulated for an $n \times m$ S-box as an n-ary global Boolean constraint in the following way:

$$\bigwedge_{i=0}^{2^{n-1}-1} f(x_i \oplus x_{2^n-i-1}) \leq 8 \tag{5.1}$$

Example 5.1 *Consider the S-box S_8 of DES of Figure 1.1, repeated in Figure 5.1 for convenience.*

Inputs 0 and 63 differ by all 6 bits, so do inputs 1 and 62, 2 and 61, and in general, i and $(63 - i)$, $0 \leq i < 32$. We are interested in corresponding outputs of x_i and x_{63-i}. There are 32 pairs of outputs, (x_i, x_{63-i}). Their exclusive-OR is equal to $x_i \oplus x_{63-i}$ which is a 4-bit value, ranging from 0 up to 15. Let $d \equiv d(x_i, x_{63-i}) = x_i \oplus x_{63-i}$ represent these 4-bit values.

Listed in Table 5.1 are the pairs (x_i, x_{63-i}) and the distance between the members of each pair. For example, $(x_0, x_{63}) = (13, 11) = (1101_2, 1011_2)$, and $d(x_0, x_{63}) = x_0 \oplus x_{63} = 1101_2 \oplus 1011_2 = 0110_2 = 6$, shown in the second and third columns for the first row of the table. There are 32 such rows for the 32 pairs.

The number of times the distances d occurs is now summarized in Table 5.2.

(x_i, x_{63-i}) $0 \leq i < 32$	**Corresponding** S**-box values**	**Distances/Differences** $d(x_i, x_{63-i}) \equiv d = x_i \oplus x_{63-i}$
(x_0, x_{63})	(13, 11)	6
(x_1, x_{62})	(1,8)	9
(x_2, x_{61})	(2,6)	4
(x_3, x_{60})	(15,5)	10
(x_4, x_{59})	(8,5)	13
(x_5, x_{58})	(13,3)	14
(x_6, x_{57})	(4,3)	7
(x_7, x_{56})	(8,15)	7
(x_8, x_{55})	(6,0)	6
(x_9, x_{54})	(10,13)	7
(x_{10}, x_{53})	(15,9)	6
(x_{11}, x_{52})	(3,10)	9
(x_{12}, x_{51})	(11,12)	7
(x_{13}, x_{50})	(7,6)	1
(x_{14}, x_{49})	(1,15)	14
(x_{15}, x_{48})	(4,0)	4
(x_{16}, x_{47})	(10,13)	7
(x_{17}, x_{46})	(12,2)	14
(x_{18}, x_{45})	(9,8)	1
(x_{19}, x_{44})	(5,14)	11
(x_{20}, x_{43})	(3,10)	9
(x_{21}, x_{42})	(6,12)	10
(x_{22}, x_{41})	(14,4)	10
(x_{23}, x_{40})	(11,9)	2
(x_{24}, x_{39})	(5,7)	2
(x_{25}, x_{38})	(0,1)	1
(x_{26}, x_{37})	(0,14)	14
(x_{27}, x_{36})	(14,4)	10
(x_{28}, x_{35})	(12,1)	13
(x_{29}, x_{34})	(9,11)	2
(x_{30}, x_{33})	(7,2)	5
(x_{31}, x_{32})	(2,7)	5

Table 5.1: Pairs of output bits for DES S-box S_8, whose corresponding input bits differ by all 6 bits, along with differences (distances) between these output-pairs.

Distances, d	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Frequency, $f(d)$	0	3	3	0	2	2	3	5	0	3	4	1	0	2	4	0

Table 5.2: Frequency of occurrence of d of Table 5.1

For example, the value $d = 7$ occurs five times because in each of the five pairs (x_6, x_{57}), (x_7, x_{56}), (x_9, x_{54}), (x_{12}, x_{51}) and $x_{16}, x_{47})$, the first value is at a distance 7 from the second.

None of the frequencies in Table 5.2 exceeds 8 and therefore DES S-box S_8 satisfies criterion **S-7**.

If any frequency in the table exceeds 8 for an S-box, that S-box violates criterion **S-7** *and should be rejected.*

Remark 5.1 *After creating the frequency table for the entire S-box, the following equation holds:*

$$\sum_{d=0}^{2^n-1} f(d) = \frac{|X|}{2}$$

Function	**Description**
`InitPartialVarsAndCounts()`	**if** $(k = 0)$ **then** $c_0, c_1, \ldots, c_{2^m-1} \leftarrow 0$
`CheckPartialSBox()`	return **true**
`ProcessNary()`	return $(k = \|X\|)$.
`MakeAndCheckSBox()`	$\Phi \leftarrow$ `MakeSBox`(X) Compute $\sigma_X(\Phi)$ using Equation 4.6 Let $d \leftarrow x_i \oplus x_{2^m-1-i}, 0 \leq i < 2^{m-1}$ Compute $c_d = f(d)$ as in Table 5.2 if $(\sigma_X(\Phi) \leq \tau) \wedge ((\forall d)(c_d \leq 8))$ return Φ
`ProcessOtherDomains()`	return **true**
`ReverseUpdateCountDistSet()`	(No operation)
`ReverseUpdateCount()`	(No operation)

Table 5.3: Functions for Nonincremental, Complete Heuristics for **S-2** and **S-7**, in `Solver` (Section 2.7)

5.2 A Non-incremental Heuristic for the COUNT constraint

The first heuristic is a straightforward implementation of Eq. 5.1, as Table 5.3 suggests. This Table is an extension of Table 4.5, with the check for Eq. 5.1 amplified. We will denote this implementation by $H_S^{\phi,\tau}$.

Alternatively, we could extend the implementation of Table 4.5 to include this heuristic for **S-7**, which we will call $H_C^{\phi,\tau}$. In either case, one can expect this implementation of the COUNT constraint to be inefficient (systematic generate-and-test), and an improvement upon S-box search speed is necessary.

5.3 An Incremental Heuristic that checks Partially-Assigned S-boxes for the COUNT Constraint

For a completely-filled S-box Φ having ϕ variables, instead of incrementing counts and checking for **S-7** after instantiating all ϕ variables the way the non-incremental heuristic does, a partial S-box Φ' that eventually extends to Φ is considered. 2^m counts $\{c_0, c_1, \ldots, c_{2^m-1}\}$ are initialized to zero. Each of these counts is subscripted by the difference $d = x_i \oplus x_{2^n-1-i}$ between the outputs x_i and x_{2^n-1-i} of Φ whose inputs differ by all m bits. The counts are incremented by 1 after at least $\frac{|X|}{2}$ assignments are made. If any count c_d exceeds 8, **S-7** is violated and the solver rejects the partial S-box Φ'. Table 5.5 implements this heuristic, extending Table 4.8. We will denote this implementation by $H_{C_7}^{\phi,\tau}$.

Example 5.2 *Consider the partially assigned S-box of Figure 5.2 having variables x_0 up to x_{40} assigned. Since variables up to x_{31} are assigned, the frequencies c_d can be determined starting from the assignment to x_{32}. These frequencies are recorded in Table 5.4.*

0	3	5	6	9	10	12	15	7	4	2	1	14	13	11	8
3	0	6	5	10	9	15	12	4	7	1	2	13	14	8	11
3	0	6	5	10											
0	3	5	8												

Figure 5.2: A Partial S-box obtained by assigning values to variables $x_0, x_1, \ldots, x_{40}$

Note that $x_{22} = 1$ from Figure 5.2. As a result of earlier domain-reductions due to AC2001 following earlier assignments, the (reduced) domain of x_{41} is $\{6, 9, 12, 15\}$. Consider the assignment $x_{41} = 6$. Let $d = x_{41} \oplus x_{63-41} = x_{41} \oplus x_{22} =$

Distances, d	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Frequency, c_d	0	0	0	0	0	0	1	0	8	0	0	0	0	0	0	0

Table 5.4: Frequency of occurrence of d for the Partially Assigned S-box of Figure 5.2

$6 \oplus 1 = 7$. *Since* $c_d = c_7 = 0 \neq 8$, *increment* c_7 *by 1 so that the entry for* c_7, *which is 0 in Table 5.4, now becomes 1 and the next variable* x_{42} *is considered for assignment. The process repeats.*

Consider now the assignment $x_{41} = 9$. *Let* $d = x_{41} \oplus x_{63-41} = x_{41} \oplus x_{22} = 9 \oplus 1 = 8$. *Note that* $c_d = c_8 = 8$ *from Table 5.4, and therefore the assignment* $x_{41} = 9$ *increases* c_8 *by 1 to 9, violating the constraint for* **S-7**. *This assignment therefore results in rejection of the resulting partially assigned* S*-box and* $x_{41} = 9$ *has to be abandoned.*

Remark 5.2 *If* $\frac{|X|}{2} + k$ *variables are assigned values, then the following is true:*

$$\sum_{d=0}^{2^n-1} c_d = k$$

If $k = \frac{|X|}{2}$, then this equation reduces to the one mentioned in Remark 5.1.

Remark 5.3 *An improvement to* $H_{C_7}^{\phi,\tau}$ *can be made by reducing the number of checks for* **S-7**. *Checking if* $c_d > 8$ *immediately after assigning values to the first* $\frac{|X|}{2}$ *variables is of no use since all* c_d*'s except one are set to zero and the one* c_d *will be equal to 1. No* c_d *will ever attain 8 at this point. The starting point to verify if* $c_d > 8$ *is, actually, after* $\frac{|X|}{2} + 8$ *variables are assigned values since, at this point it is possible that exactly one* c_d *attained 8 for some* d *while all others are zeros.* ($k = 8$ *in Remark 5.2.)*

$H_{C_7}^{\phi,\tau}$ can be further improved by a novel approach of integrating the n-ary constraint into the solver, projecting onto domains of future variables reducing these during the process, prior to applying AC2001. The approach is discussed next.

5.4 An Incremental Heuristic that employs Projections

Just as in the earlier incremental heuristic for **S-7**, let c_d denote the count of distances d where d is the distance between x_i and x_{2^n-i-1} representing S-box

Function	**Description**
`InitPartialVarsAndCounts()`	Let $X' \leftarrow \{x : x \in X \wedge x \text{ is assigned.}\}$ **if** $(k = 0)$ **then** Let: $c_0, c_1, \ldots, c_{2^m-1} \leftarrow 0$

`CheckPartialSBox()`	Set `s2ok` $\leftarrow$ **true**, `s7ok` $\leftarrow$ **true**
	if $(k \geq \frac{\lvert X\rvert}{2} + \tau)$ **then** {Check **S-2** partially}
	  $\Phi' \leftarrow$ `MakePartialSBox`(X')
	  if Φ' violates either Property 4.3 or Property 4.4 **then** `s2ok` $\leftarrow$ **false**
	end if
	if $(k \geq \frac{X}{2}) \wedge$ `s2ok` **then** {Check **S-7** partially}
	  Let $d \leftarrow x_k \oplus x_{\lvert X\rvert-k-1}$
	  if $(c_d + 1 > 8)$ **then** `s7ok` $\leftarrow$ **false**
	  if `s7ok` **then** $c_d \leftarrow c_d + 1$
	end if
	return `s2ok` $\wedge$ `s7ok`

`ProcessNary()`	**if** $(k = \lvert X\rvert)$ **then** return **true** **else** return **false**
`MakeAndCheckSBox()`	$\Phi \leftarrow$ `MakeSBox`(X)
`ProcessOtherDomains()`	return **true**
`ReverseUpdateCountDistSet()`	(No operation)
`ReverseUpdateCount()`	**if** $(k \geq \frac{\lvert X\rvert}{2})$ **then** $c_d \leftarrow c_d - 1$

Table 5.5: Functions for $H_{C_7}^{\phi;\tau}$ in `Solver` (Section 2.7)

outputs for those pairs of inputs that differ by all n bits, with $0 \leq d < 2^m$. All the c_d's, namely, $c_0, c_1, \ldots, c_{2^m-1}$, are initialized to zero. So long as $|X'| < \frac{|X|}{2}$, c_d is not updated, similar to what occurs in function `CheckPartialSBox` of heuristic $H_{C_7}^{\phi;\tau}$.

For assignments to the next eight variables x_i when $\frac{|X|}{2} \leq i < \frac{|X|}{2}+8$, increment the count $c_d = c_{|X|-i-1}$. During these assignments, $1 \leq c_d < 8$ and no checking for $c_d > 8$ is needed at this point.

The checking for **S-7** begins when c_d becomes equal to 8 starting from the next variable assignment. Denote this next assignment by $x_k = v$, with $\frac{|X|}{2} + 8 \leq k < |X|$ and $v \in D'_k$, D'_k being the (reduced) domain of x_k. A *distance-set* $\Delta = \{d = x_{|X|-k-1} \oplus x_k : c_d = 8\}$ is formed from this point onwards, with $0 \leq |\Delta| \leq 2^m$.

Whenever $\frac{|X|}{2} + 8 \leq k < |X|$, remove values $f \in \{x_{|X|-k-1} \oplus e : e \in \Delta\}$ from the domains in $D' \setminus D'_{X'} \setminus \{D'_k\}$ of all future variables. If a domain becomes empty due to this removal, abandon the assignment $x_k = v$ since a dead-end is encountered. Otherwise, add each of these removed values f to the deletion set DS.

This heuristic also needs to restore values of counts c_d and also, undoes the current addition to the distance-set Δ at the current level of recursion before termination. Table 5.6 implements this heuristic, and is an extension of Table 4.8. We will denote the implementation of Table 5.6 by $H^{\phi;\tau}_{AC7}$.

Remark 5.4 *$H^{\phi;\tau}_{AC7}$ ensures that $0 \leq c_d < 9$.*

For this reason, no explicit checking of **S-7** (incrementally) is required as is the case with $H^{\phi;\tau}_{C_7}$.

Property 5.1 *$H^{\phi;\tau}_{AC7}$ gives the same set of S-boxes as $H^{\phi;\tau}_{C_7}$.*

Proof

Let $d = v \oplus w$ and $c_d = 8$. Consider the partial S-box having variable x_k that is to be assigned the value v from its domain D (i.e. $v \in D$), and having a variable $x_{|X|-k-1} = w$ already assigned, where $\frac{|X|}{2} + 8 \leq k < |X|$. It is enough to prove that both heuristics $H^{\phi;\tau}_{C_7}$ and $H^{\phi;\tau}_{AC7}$ will not assign $x_k = v$.

Suppose $H^{\phi;\tau}_{C_7}$ tentatively assigns $x_k = v$. Then we have

$$x_k \oplus x_{|X|-k-1} = v \oplus w = d$$

and $H^{\phi;\tau}_{C_7}$ increments c_d by 1, so that

$$c_d > 8$$

.

violating **S-7**. $H^{\phi;\tau}_{C_7}$ abandons the assignment $x_k = v$ which will not be considered.

Since $c_d = 8$, $H^{\phi;\tau}_{AC7}$ has included d in the distance-set Δ so that $d \in \Delta$. To project $x_{|X|-k-1}$ on D that is the domain of future variable x_k, $H^{\phi;\tau}_{AC7}$ now traverses Δ to compute

$$e = d \oplus x_{|X|-k-1} = (v \oplus w) \oplus w = v$$

Function	**Description**
`InitPartialVarsAndCounts()`	Let $X' \leftarrow \{x : x \in X \wedge x \text{ is assigned.}\}$ Let $\Delta \leftarrow \phi$ **if** $(k = 0)$ **then** $c_0, c_1, \ldots, c_{2^m-1} \leftarrow 0$

`CheckPartialSBox()`	Set `s2ok` $\leftarrow$ **true**
	if $(k \geq \frac{\lvert X\rvert}{2} + \tau)$ **then** {Check **S-2** partially}
	  $\Phi' \leftarrow$ `MakePartialSBox`(X')
	  if Φ' violates either Property 4.3 or Property 4.4 **then** `s2ok` $\leftarrow$ **false**
	end if
	return `s2ok`

`ProcessNary()`	**if** $(k = \lvert X\rvert)$ **then** return **true else** return **false**
`MakeAndCheckSBox()`	$\Phi \leftarrow$ `MakeSBox`(X)

`ProcessOtherDomains()`	**if** $k \geq \frac{\lvert X\rvert}{2}$ **then**
	  Set $d \leftarrow x_j \oplus x_{\lvert X\rvert-j-1}$, $c_d \leftarrow c_d + 1$
	  if $(k \geq \frac{\lvert X\rvert}{2} + 8) \wedge (c_d = 8)$ **then**
	    $\Delta \leftarrow \Delta \cup \{d\}$
	    for all $e \in \Delta$ **do**
	      Set $f \leftarrow x_{\lvert X\rvert-j-1} \oplus e$
	      for all $x \in X \setminus (X' \cup \{x_j\})$ **do**
	        if $f \in D'_x$ **then**
	          $D'_x \leftarrow D'_x \setminus \{f\}$
	          if $(D'_x = \phi)$ return **false**
	          $DS \leftarrow DS \cup \{f\}$
	        end if
	      end for
	    end for
	  end if
	end if
	return **true**

Function	Description

`ReverseUpdateCountDistSet()`	**if** $(k \geq \frac{\lvert X \rvert}{2})$ **then**
	  if $(k \geq \frac{\lvert X \rvert}{2} + 8) \wedge (c_d = 8)$ **then**
	    Set $\Delta \leftarrow \Delta \setminus \{d\}$
	  end if
	  Set $c_d \leftarrow c_d - 1$
	end if

`ReverseUpdateCount()`	(No operation)

Table 5.6: Functions for Heuristic $H^{\phi;\tau}_{AC7}$ in `Solver` (Section 2.7)

.

Since $v \in D$, H_{AC7} removes v from D, avoiding the assignment $x_k = v$.

Thus heuristics $H^{\phi;\tau}_{C_7}$ and $H^{\phi;\tau}_{AC7}$ do not assign $x_k = v$ and this completes the proof.

Q.E.D.

Example 5.3 *Consider the partially assigned S-box of Figure 5.3 having variables x_0 up to x_{47} assigned. Since variables up to x_{31} are assigned, the frequencies c_d can be determined starting from the assignment to x_{32}. These frequencies are recorded in Table 5.7.*

0	3	5	6	9	10	12	15	7	4	2	1	14	13	11	8
3	0	6	5	10	9	15	12	4	7	1	2	13	14	8	11
3	0	6	5	15	12	10	9								
0	3	5	6	12	15	9	10								

Figure 5.3: A Partial S-box obtained by assigning values to variables $x_0, x_1, \ldots, x_{47}$

From the frequency table of Table 5.7, two distances d have counts equal to 8, namely, when $d = 8, 13$. These two values of d form the distance-set $\Delta = \{8, 13\}$. As a result of earlier domain-reductions due to AC2001 following earlier assignments, the (reduced) domains for the future variables $x_{48}, x_{49}, \ldots, x_{63}$ are recorded in the second column of Table 5.8.

Now traverse each element $e \in \Delta$ and remove values $f = e \oplus x_{63-i}$ from the domain of x_i, $48 \leq i < 64$. The computed values for f are in the third column

Distances, d	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Frequency, c_d	0	0	0	0	0	0	0	0	8	0	0	0	0	8	0	0

Table 5.7: Frequency of occurrence of d for the Partially Assigned S-box of Figure 5.3

of Table 5.8, while the reduced domains of the future variables are in the last column of this table. For future variable x_{56} *having (reduced) domain* $\{1, 2, 4, 8\}$, $x_{63-56} = x_7 = 5$ *from Figure 5.3.* $e \in \{8, 13\}$ *and* $f \in \{8 \oplus 5, 13 \oplus 5\} = \{13, 8\}$ *as shown in the third column of Table 5.8 against variable* x_{56}. *Removing these values from the domain of* x_{56} *results in its reduced domain of* $\{1, 2, 4\}$.

Future Variable	**Domain before Value Removal**	**Values to Remove**	**Domain After Value Removal**
x_{48}	$\{8, 13, 14\}$	$\{4, 1\}$	$\{8, 13, 14\}$
x_{49}	$\{11, 13, 14\}$	$\{7, 2\}$	$\{11, 13, 14\}$
x_{50}	$\{11, 13, 14\}$	$\{7, 2\}$	$\{11, 13, 14\}$
x_{51}	$\{8, 13, 14\}$	$\{4, 1\}$	$\{8, 13, 14\}$
x_{52}	$\{8, 11, 13\}$	$\{1, 4\}$	$\{8, 11, 13\}$
x_{53}	$\{8, 11, 14\}$	$\{2, 7\}$	$\{8, 11, 14\}$
x_{54}	$\{8, 11, 14\}$	$\{2, 7\}$	$\{8, 11, 14\}$
x_{55}	$\{8, 11, 13\}$	$\{1, 4\}$	$\{8, 11, 13\}$
x_{56}	$\{1, 2, 4, 8\}$	$\{13, 8\}$	$\{1, 2, 4\}$
x_{57}	$\{1, 2, 7, 11\}$	$\{14, 11\}$	$\{1, 2, 7\}$
x_{58}	$\{1, 2, 7, 11\}$	$\{14, 11\}$	$\{1, 2, 7\}$
x_{59}	$\{1, 2, 4, 8\}$	$\{13, 8\}$	$\{1, 2, 4\}$
x_{60}	$\{1, 4, 7, 13\}$	$\{8, 13\}$	$\{1, 4, 7\}$
x_{61}	$\{2, 4, 7, 14\}$	$\{11, 14\}$	$\{2, 4, 7\}$
x_{62}	$\{2, 4, 7, 14\}$	$\{11, 14\}$	$\{2, 4, 7\}$
x_{63}	$\{1, 4, 7, 13\}$	$\{8, 13\}$	$\{1, 4, 7\}$

Table 5.8: Domains of future variables with values being removed due to projections of past assignments

5.5 Heuristics

We have discussed the following heuristics employed by `Solver` for the n-ary constraints in our model, as governed by functions `InitPartialVarsAndCounts`, `PreProcess`, `CheckPartialSBox`, `ProcessNary`, `MakeAndCheckSBox`, `ProcessOtherDomains`, `ReverseUpdateCountDistSet` and `ReverseUpdateCount` (Section 2.7).

Heuristic $H_S^{\phi,\tau}$ Check the criteria **S-2** and **S-7** after a complete S-box is formed.

Heuristic $H_C^{\phi,\tau}$ Decompose the n-ary soft constraint **S-2** into hard constraints by projection onto already-assigned variables $X' \subseteq X$ (Chapter 5). Check **S-7** after an entire S-box is formed, rejecting if **S-7** is violated.

Heuristic $H_{C_7}^{\phi,\tau}$ Decompose the n-ary soft constraint **S-2** into hard constraints by projection onto already-assigned variables $X' \subseteq X$ (Chapter 5). Check **S-7** at each assignment where applicable.

Heuristic $H_{AC7}^{\phi,\tau}$ Decompose the n-ary soft constraint **S-2** into hard constraints by projection onto already-assigned variables $X' \subseteq X$ (Chapter 5). Project the not-so-straightforwardly-decomposable n-ary constraint **S-7** into binary constraints.

We now introduce an optimization to `Solver` (Section 2.7) to reduce calls to function `EstablishAC`, and subsequently discuss a variable-ordering heuristic as governed by function π.

5.5.1 Optimizations on the three Heuristics

In `Solver` (Section 2.7), the deletion set DS is always populated by adding to it all values other than the one being assigned to the current variable, from its (reduced) domain (Line 12). In case this domain is a singleton, DS is empty and no arc-consistency check is required. Accordingly, the call to function `EstablishFullAC` can be made conditional. The program segment after Line 16 in `Solver` takes the form shown in Table 5.9. This optimization reduces the number of calls to the function `EstablishFullAC` resulting in some speedup as the experiments will reveal.

We refer to the three heuristics respectively, as $HO_S^{\phi,\tau}$, $HO_{C_7}^{\phi,\tau}$ and $HO_{AC7}^{\phi,\tau}$ with the optimization introduced in `Solver()`. Chapter 7 illustrates the speedup of solution generation resulting from this optimization and reduction of arc-consistency checks resulting from this optimization.

```
**if** $c_4$ **then**
   **if** $DS \neq \phi$ **then**
      $D'' \leftarrow$ EstablishAC$(D', DS)$
   **else**
      $D'' \leftarrow D'$
   **end if**
   **if** no domain in $D''$ is empty **then**
      Solver$(X, D'', C_2, k+1, \tau)$
   **end if**
**end if**
ReverseUpdateCountDistSet()
```

Table 5.9: An optimization introduced in `Solver` (Section 2.7)

5.5.2 Variable Ordering Heuristics

In the three heuristics $H_S^{\phi;\tau}$, $H_{C_7}^{\phi;\tau}$ and $H_{AC7}^{\phi;\tau}$, and their optimized variants $HO_S^{\phi;\tau}$, $HO_{C_7}^{\phi;\tau}$ and $HO_{AC7}^{\phi;\tau}$ respectively, variable-ordering heuristics are employed as specified in the function `SelectNextVariable` of `Solver` (Section 2.7). We will consider two such heuristics.

Straight-Line Variable-Ordering Heuristic In a *Straight-Line Variable-Ordering Heuristic*, the variables are assigned in the logical order while reading S-box entries, that is, row-wise, for the S-box arrangement of Figure 3.1. Referring to this figure, the sequence of variable ordering is $x_0, x_2, \ldots, x_{30}, x_1, x_3, \ldots, x_{31}, x_{32}, x_{34}, \ldots, x_{62}, x_{33}, x_{35}, \ldots, x_{63}$. For this heuristic, the ordering function π assumes the following definition for an $n \times m$ S-box:

$$\pi(k) = \begin{cases} 2k, & \text{if } k < 2^{n-2} \\ 2k - 2^{n-1} + 1, & \text{if } 2^{n-2} \leq k < 2^{n-1} \\ 2k - 2^{n-1}, & \text{if } 2^{n-1} \leq k < 3 \times 2^{n-2} \\ 2k - 2^n + 1 & \text{if } 3 \times 2^{n-2} \leq k < 2^n \end{cases} \tag{5.2}$$

Example 4.2 demonstrates a partially assigned S-box that follows the straight-line variable ordering heuristic.

Zig-Zag Variable-Ordering Heuristic A second heuristic under consideration is what we will refer to as the *Zig-Zag Variable Ordering Heuristic*. Here, variables are assigned values in a zig-zag order, that is, $x_0, x_1, x_2, \ldots, x_{63}$. If one visits the variables in the S-box of Figure 3.1, a zig-zag pattern is observed and hence the name for this heuristic. For this heuristic, the ordering function π assumes the following definition for the sequence of variables of an $n \times m$ S-box:

$$\pi(k) = k \tag{5.3}$$

This is the variable-ordering heuristic that has been used in all the examples of this Chapter. In particular, example 4.1 demonstrates a partially assigned S-box that follows the zig-zag variable ordering heuristic.

The definitions of $\pi(k)$ given by equations 5.2 and 5.3 are to be used in `Solver` (Section 2.7), in the function `SelectNextVariable(k)`.

Analogs of the three heuristics $H_S^{\phi,\tau}$, $H_{C_7}^{\phi,\tau}$ and $H_{AC7}^{\phi,\tau}$, and their optimized variants $HO_S^{\phi,\tau}$, $HO_{C_7}^{\phi,\tau}$ and $HO_{AC7}^{\phi,\tau}$ respectively, now result. These heuristics are called $V_S^{\phi,\tau}$, $V_{C_7}^{\phi,\tau}$ and $V_{AC7}^{\phi,\tau}$, and their optimized variants, $VO_S^{\phi,\tau}$, $VO_{C_7}^{\phi,\tau}$ and $VO_{AC7}^{\phi,\tau}$, respectively, for Straight-Line variable ordering. For Zig-Zag variable ordering, the heuristics will have the same annotation, namely, $H_S^{\phi,\tau}$, $H_{C_7}^{\phi,\tau}$, $H_{AC7}^{\phi,\tau}$, and their optimized variants $HO_S^{\phi,\tau}$, $HO_{C_7}^{\phi,\tau}$ and $HO_{AC7}^{\phi,\tau}$, respectively.

A step should not be present in heuristic $V_{AC7}^{\phi,\tau}$ ($VO_{AC7}^{\phi,\tau}$) that is present in $H_{AC7}^{\phi,\tau}$ ($HO_{AC7}^{\phi,\tau}$). In Table 5.6, the extra check ($k \geq \frac{|X|}{2} + 8$) that appears in functions `ProcessOtherDomains` and `ReverseUpdateCountDistSet` for heuristic $H_{AC7}^{\phi,\tau}$ ($HO_{AC7}^{\phi,\tau}$) is no longer applicable, and this check should be removed from these two places while implementing $V_{AC7}^{\phi,\tau}$ ($VO_{AC7}^{\phi,\tau}$).

5.5.3 Domain Ordering Heuristics

The ordering of domains for each variable is another consideration worth exploring from the viewpoint of efficiency of search. No modification is done to the solver and no new steps are necessary. Only the domains precompiled into the solver are permuted. Two domain-ordering heuristics are examined.

Default Ordering In this heuristic, the domain for each variable is ordered in ascending order, that is, for each variable of an $n \times m$ S-box, its domain is the set $D = \{0, 1, 2, \ldots, 2^m - 1\}$.

Random Permutation Here the domain of each variable is randomly permuted based on a specified seed. In the section on experiments, we will consider one such random ordering for the purposes of comparison.

Further discussion on domain-ordering heuristics is made in Chapter 7.

5.6 Summary and Looking Ahead

The COUNT constraint is a global n-ary constraint. We have discussed a non-incremental approach to checking for this constraint, and an incremental approach to checking on partially assigned S-boxes.

The global n-ary constraint for criterion **S-7** is not straightforwardly decomposable into binary constraints. Nevertheless, we are able to integrate this constraint into the solver by projecting past assignments onto the domains of future variables. During the process, these domains are reduced.

An optimization has been introduced in `Solver` (subsection 5.5.1) to avoid unnecessary arc-consistency checking in case nothing is added to the deletion set DS. This happens when the (reduced) domain of the variable being considered for assignment contains only one element.

Two kinds of variable ordering heuristics are considered (subsection 5.5.2): A straight-line variable ordering and an alternative zig-zag variable ordering. Value-ordering is considered by randomly permutating domains (subsection 5.5.3).

Chapter 7 discusses the performances of the several heuristics under these circumstances.

Chapter 6

Symmetry. The Multiple S-box Problem

In this chapter, we will discuss symmetry of S-boxes strictly from a CSP standpoint. Our main objective in this Chapter is to demonstrate new forms of symmetries in S-boxes that the CSP model so elegantly exposes.

We also discuss the criterion **S-8** that involves arrangements of S-boxes taken eight at a time, and selection of arrangements that minimizes a cost function.

6.1 Symmetry in CSPs and Symmetric Constraints

A constraint is *symmetric* if, upon interchanging some or all of the variables (values to each variable) with others in its scope (domain), the constraint is not altered.

Example 6.1 *The ternary constraint given by the equation $x + y + z = 5$, is symmetric, because, if we interchange x and y, the resulting constraint becomes $y + x + z = 5$ which is the same as the given one. Similar is the case if x is interchanged with y, y with z and z with x simultaneously. This is an example of a constraint with* variable *symmetry.*

6.1.1 Symmetry and Efficiency

Due to symmetry, solution tuples that are encountered far later in the systematic search for solutions would already have been available if the symmetry were identified, adding to efficiency of solution generation.

Example 6.2 *Let the domains of the variables x, y and z of Example 6.1 be the set of integers $\{0, 1, 2, 3, 4, 5\}$. The assignment $x = 0, y = 0, z = 5$, or $(0, 0, 5)$ for brevity, clearly satisfies the constraint $x + y + z = 5$. Upon simultaneously replacing x, y, z with z, y, x respectively, the resulting constraint is unchanged as Example 6.1 suggests. Now the abbreviated assignment becomes $(5, 0, 0)$ which is also a solution. This solution would have occurred much later in the solution space after a systematic search. But we are able to identify the same immediately due to symmetry. In other words, identifying symmetry adds to efficiency of solution generation.*

6.1.2 Value, Variable and Conditional Symmetry

A constraint possesses value symmetry if interchanging valid values assigned to variables continues to satisfy the constraint. It possesses variable symmetry if the variables in its scope can be permuted, and conditional symmetry if the solutions are preserved subject to some condition. A CSP can possess more than one of these forms of symmetry simultaneously. For example, the constraints analyzed in Examples 6.1 and 6.2 possess both, variable and value symmetry.

Example 6.3 *Consider the following CSP for the problem of finding Pythagorean Triplets, involving variables $X = \{x, y, z\}$:*

$$\begin{aligned} x^2 + y^2 &= z^2 \\ x, y, z &\in \{3, 4, \ldots, 100\} \end{aligned}$$

A solution is $x = 3, y = 4, z = 5$. Upon interchanging the values of x and y, the assignment $x = 4, y = 3, z = 5$ is also a solution. This CSP possesses value symmetry. Moreover, if x and y are interchanged, the equation remains mathematically unaltered and the CSP possesses variable symmetry as well.

6.1.3 Breaking of Symmetry in CSPs

Breaking of all symmetries is shown to be NP-hard, however, there are practical ways by which most of these can be broken [70]. For example, constraints are added to remove the aforementioned forms of symmetry [48, 69].

Example 6.4 *Consider Example 6.3 that involves finding Pythagorean Triplets. A number of solutions involve values of x and y interchanged due to symmetry, such as, $x = 5, y = 12, z = 13$ and $x = 12, y = 5, z = 13$. It is often sufficient to have one of these two solutions instead of having both. By imposing an ordering on the values of x and y, namely, by adding the constraint,*

$$x < y$$

the solution $x = 12, y = 5, z = 13$ is eliminated and symmetry is broken.

6.2 Symmetry in S-boxes

Using our CSP model formulated in Chapters 3, 4 and 5, we have identified five forms of symmetry in S-boxes directly arising out of the criteria for DES **S-2** to **S-7**. These forms are the following:

1. Row Symmetry, due to criterion **S-4**
2. Column Symmetry, due to criterion **S-5**
3. Diagonal Symmetry, due to criterion **S-6**,
4. Rotational Symmetry, due to criterion **S-7**, and
5. Bit Inversion Symmetry, due to criteria **S-2**, **S-3**, **S-4**, **S-5**, **S-6** and **S-7**.

The granularity of the symmetry being addressed is very important. We have symmetry of an individual constraint, we have symmetry of sets of constraints, and ultimately of the entire CSP. An individual constraint may exhibit symmetry but the appropriate transformation may violate another constraint and therefore, may not result in an S-box. Forms of symmetry that satisfy all constraints and therefore, result in newer S-boxes, are also prevalent. The Row, Column and Diagonal symmetry fall under the first category, not necessarily resulting in S-boxes. On the other hand, the Rotational and Bit Inversion symmetry will satisfy all the other constraints, and will yield alternative S-boxes, as we will see in this Chapter.

6.3 Relevant Properties of the exclusive-OR Operator, Parity, and the Linear Approximation Table

Let us first discuss properties of the exclusive-OR operator relevant to symmetry of S-boxes. We will prove these for two bits a and b and extend these to the general case where the operands are n-bit quantities. Next, we discuss the parity of a number and its properties. Finally, we discuss properties of the Linear Approximation Table for a 6×4 S-box that will be used in Rotational and Bit Inversion symmetries.

6.3.1 Relevant Properties of the exclusive-OR Operator

Definition 6.1 *The exclusive-OR operator on two Boolean quantities a and b is defined by the following expression:*

$$a \oplus b = \overline{a} \cdot b + a \cdot \overline{b} \tag{6.1}$$

Remark 6.1 *The exclusive-OR operator on two n-bit quantities $A = a_0a_1 \ldots a_{n-1}$ and $B = b_0b_1 \ldots b_{n-1}$ is equal to the n-bit bit-pattern formed by the bits $a_0 \oplus b_0$, $a_1 \oplus b_1$, $\ldots$, $a_{n-1} \oplus b_{n-1}$. In other words, if n is the word-length of a computer, the operation $A \oplus B$ results in n parallel exclusive-OR operations on 1-bit operands.*

From this definition, it is easy to deduce the following properties for two Boolean entities a and b. Generalizations to n-bit bit-patterns A and B are easily made.

1. $a \oplus 0 = 0 \oplus a = a$

2. $a \oplus 1 = 1 \oplus a = \overline{a}$

3. $a \oplus a = 0$

4. $a \oplus \overline{a} = 1$

5. The exclusive-OR operation on two Boolean quantities a and b is commutative, that is, $a \oplus b = b \oplus a$.

6. The exclusive-OR operation on three Boolean quantities a, b and c is associative, that is, $(a \oplus b) \oplus c = a \oplus (b \oplus c)$.

7. The result of the exclusive-OR operation on a and b is the same as that of their one's-complements, $\overline{a}$ and $\overline{b}$ respectively, that is, $\overline{a} \oplus \overline{b} = a \oplus b$.

8. For Boolean variables a and b, the following equations hold: $a \oplus (a \oplus b) = b$, $b \oplus (a \oplus b) = a$. This property easily extends to two n-bit operands A and B, that is, $A \oplus (A \oplus B) = B$ and $B \oplus (A \oplus B) = A$. The idea is used in stream ciphers for encryption and decryption. It is also used in high-speed software implementations of symmetric key cryptographic algorithms.

6.3.2 The Parity of a Bit Pattern

Recall that the parity of an n-bit bit pattern $a = a_0 a_1 \ldots a_{n-1}$ is equal to the number of 1's in the bit pattern. If there are an odd number of 1's, then we say that a is of *odd parity*; if this count is an even number, a is said to be of *even parity*. The parity p of a is also equal to the exclusive-OR of the bits in a, that is,

$$p = a_0 \oplus a_1 \oplus \ldots \oplus a_{n-1}$$

Property 6.1 *If an n-bit number a has parity p ($p = 1$ for odd parity and 0 for even), then its 1's-complement $\bar{a}$ has parity $\bar{p}$ if n is odd and p if n is even.*

Proof Let $a = a_0a_1a_2\ldots a_{n-1}$. Then by the definition of parity,

$$p = a_0 \oplus a_1 \oplus a_2 \oplus \ldots \oplus a_{n-1} \tag{6.2}$$

We also have the following Boolean identity:

$$1 \oplus 1 \oplus \ldots \oplus 1,\ n \text{ times} = \begin{cases} 1, & \text{for odd } n \\ 0, & \text{for even } n \end{cases} \tag{6.3}$$

Let q be the parity of $\bar{a}$, obtained by exclusive-OR of each bit in a with 1. Then,

$$\begin{aligned} q &= (a_0 \oplus 1) \oplus (a_1 \oplus 1) \oplus \ldots \oplus (a_{n-1} \oplus 1) \\ &= (a_0 \oplus a_1 \oplus a_2 \oplus \ldots \oplus a_{n-1}) \oplus (1 \oplus 1 \oplus \ldots \oplus 1,\ n \text{ times}) \\ &= \begin{cases} p \oplus 1, & \text{if } n \text{ is odd} \\ p \oplus 0, & \text{if } n \text{ is even} \end{cases} \text{, using Equations 6.2 and 6.3} \\ &= \begin{cases} \bar{p}, & \text{if } n \text{ is odd} \\ p, & \text{if } n \text{ is even} \end{cases} \end{aligned}$$

Q.E.D.

6.3.3 Relevant Properties of the Linear Approximation Table

We will now prove some important results that hold for the Linear Approximation Table of an S-box of Chapter 4. For simplicity, the discussion is limited to 6×4 S-boxessince it is their symmetry that is being considered. These properties for the Linear Approximation Table are being discussed here rather than in Chapter 4, for immediate use in results on Symmetry, particularly Rotational and Bit Inversion Symmetry.

We consider one layer of the S-box Φ, obtained by making an assignment to a single variable x_i corresponding to input i (Example 4.6). Each entry in this layer is either 0 or 1 depending upon whether equation $L_a(i) = L_b(x_i)$ holds or not, for any a, b (subsection 4.2.1). In other words, we can consider each entry of a layer to be equal to $\overline{L_a(i) \oplus L_b(x_i)}$, also equal to $L_a(i) \oplus L_b(x_i) \oplus 1$.

Property 6.2 *Let $i = i_0i_1\ldots i_5$, $a = a_0a_1\ldots a_5$ and $p = a_0 \oplus a_1 \oplus \ldots \oplus a_5$ represent the parity of a, equal to the number of 1's in the binary representation of a. Then $L_a(\bar{i}) = L_a(i) \oplus p$.*

Proof We have

$$\begin{aligned} L_a(\overline{i}) &= a_0\overline{i_0} \oplus a_1\overline{i_1} \oplus \ldots \oplus a_5\overline{i_5} \\ &= a_0 \cdot (i_0 \oplus 1) \oplus a_1 \cdot (i_1 \oplus 1) \oplus \ldots \oplus a_5(i_5 \oplus 1) \\ &= (a_0 i_0 \oplus a_1 i_1 \oplus \ldots \oplus a_5 i_5) \oplus (a_0 \oplus a_1 \oplus \ldots \oplus a_5) \\ &= L_a(i) \oplus p \end{aligned}$$

which proves the proposition.

Q.E.D.

Property 6.3 *Each entry in the Linear Approximation Table corresponding to the single assignment $x_{\overline{i}} = v$ of an S-box differs from the corresponding entry for the single assignment $x_i = v$ by the parity p of a, the number of the row in the Table in which the entry is situated.*

Proof For the single assignment $x_i = v$, each entry in the Linear Approximation Table is of the form $y = L_a(i) \oplus L_b(v) \oplus 1$ as discussed at the start of this subsection. Consider the Linear Approximation Table for the assignment $x_{\overline{i}} = v$. Each entry in this table will be of the form

$$\begin{aligned} & L_a(\overline{i}) \oplus L_b(v) \oplus 1 \\ = \; & L_a(i) \oplus p \oplus L_b(v) \oplus 1, \text{ (Property 6.2)} \\ = \; & L_a(i) \oplus L_b(v) \oplus 1 \oplus p, \text{ (Commutativity of exclusive-OR)} \\ = \; & y \oplus p \end{aligned}$$

which is the required result. This result will be used in proving the invariance of the score of an S-box over Rotational symmetry.

Q.E.D.

Remark 6.2 *Property 6.3 can be reworded as follows: Given a single assignment $x_i = v$ for an S-box entry. An entry in Row a and Column b of the Linear Approximation Table for the assignment $x_i = \overline{v}$ is equal to the truth value of the following expression:*

$$L_a(i) = \begin{cases} \overline{L_b(v)}, & \textit{if } a \textit{ has odd parity} \\ L_b(v), & \textit{if } a \textit{ has even parity} \end{cases}$$

Property 6.4 *Each entry in the Linear Approximation Table corresponding to the single assignment $x_i = v$ of an S-box differs from the corresponding entry for the single assignment $x_i = \overline{v}$ by the parity q of b, the number of the column in the Table where the entry is situated.*

Proof For the single assignment $x_i = v$, each entry in the Linear Approximation Table is of the form $y = L_a(i) \oplus L_b(v) \oplus 1$ as discussed at the start of this subsection. Consider the Linear Approximation Table for the assignment $x_i = \overline{v}$. Each entry in this table will be of the form

$$\begin{aligned} & L_a(i) \oplus L_b(\overline{v}) \oplus 1 \\ = \;& L_a(i) \oplus L_b(v) \oplus q \oplus 1, \text{ (Property 6.2)} \\ = \;& L_a(i) \oplus L_b(v) \oplus 1 \oplus q, \text{ (Commutativity of exclusive-OR)} \\ = \;& y \oplus q \end{aligned}$$

which is the required result. This result will be used in proving the invariance of the score of an S-box over Bit Inversion symmetry.

Q.E.D.

Remark 6.3 *Property 6.4 can be reworded as follows: Given a single assignment $x_i = v$ for an S-box entry. An entry in Row a and Column b of the Linear Approximation Table for the assignment $x_i = \bar{v}$ is equal to the truth value of the following expression:*

$$L_a(i) = \begin{cases} \overline{L_b(v)}, & \textit{if } b \textit{ has odd parity} \\ L_b(v), & \textit{if } b \textit{ has even parity} \end{cases}$$

Let us examine each form of symmetry against the criteria specified. In the remainder of this Chapter, the discussion is made with reference to 6×4 S-boxes.

6.4 Row Symmetry

We define row symmetry of a 6×4 S-box to mean that if its rows are interchanged, the resulting configuration still satisfies the criterion (or criteria) in question.

Property 6.5 *Criterion* **S-4** *exhibits row symmetry but only if the top two and bottom two row interchanges occur simultaneously.*

Proof Consider two S-box inputs $2i$ and $(2i + 1)$ that differ only in their least significant bits, $0 \le i < 32$. The requirement of criterion **S-4** is satisfied and according to criterion **S-4**,

$$wt(x_{2i} \oplus x_{2i+1}) \ge 2 \tag{6.4}$$

By the commutativity property of the exclusive-OR operation, Equation 6.4 can be rewritten as:

$$wt(x_{2i+1} \oplus x_{2i}) \geq 2$$

This means that interchanging x_{2i+1} and x_{2i} does not affect the constraint for criterion **S-4**, suggesting that **S-4** possesses variable symmetry. With reference to Figure 6.1, variables (i.e. S-box outputs) x_{2i} occur in Row 0 whenever $0 \leq i < 16$ and Row 2 for $16 \leq i < 32$, while variables (S-box outputs) x_{2i+1} occur in the other two rows, Row 1 and Row 3. In other words, variables with even subscripts x_{2j} and corresponding variables with odd subscripts $x_{2j\oplus 1}$ having inputs that differ in their least significant bits, are in the neighboring rows of the S-box when organized as in Figure 6.1 that are interchangeable. This proves the proposition.

Q.E.D.

6.4.1 Only the Least Significant Bits

	$i_1i_2i_3i_4$															
i_0i_5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	x_0	x_2	x_4	x_6	x_8	x_{10}	x_{12}	x_{14}	x_{16}	x_{18}	x_{20}	x_{22}	x_{24}	x_{26}	x_{28}	x_{30}
1	$\mathbf{x_1}$	$\mathbf{x_3}$	$\mathbf{x_5}$	$\mathbf{x_7}$	$\mathbf{x_9}$	$\mathbf{x_{11}}$	$\mathbf{x_{13}}$	$\mathbf{x_{15}}$	$\mathbf{x_{17}}$	$\mathbf{x_{19}}$	$\mathbf{x_{21}}$	$\mathbf{x_{23}}$	$\mathbf{x_{25}}$	$\mathbf{x_{27}}$	$\mathbf{x_{29}}$	$\mathbf{x_{31}}$
2	x_{32}	x_{34}	x_{36}	x_{38}	x_{40}	x_{42}	x_{44}	x_{46}	x_{48}	x_{50}	x_{52}	x_{54}	x_{56}	x_{58}	x_{60}	x_{62}
1	$\mathbf{x_{33}}$	$\mathbf{x_{35}}$	$\mathbf{x_{37}}$	$\mathbf{x_{39}}$	$\mathbf{x_{41}}$	$\mathbf{x_{43}}$	$\mathbf{x_{45}}$	$\mathbf{x_{47}}$	$\mathbf{x_{49}}$	$\mathbf{x_{51}}$	$\mathbf{x_{53}}$	$\mathbf{x_{55}}$	$\mathbf{x_{57}}$	$\mathbf{x_{59}}$	$\mathbf{x_{61}}$	$\mathbf{x_{63}}$

Figure 6.1: **S-6** for a 6×4 S-box exhibiting Row Symmetry. The odd rows are shown in bold font.

We have considered only the least significant bits for any two input bits to the S-box. This satisfies the requirement for **S-4**, namely that the two inputs differ in only one bit. However, **S-4** need not be satisfied only in this manner. In fact, the 6-bit input i can differ by 1 bit with 6-bit inputs $i \oplus 2^j$, where $0 \leq i < 64$, $0 \leq j < 6$. That is, each variable x_i is involved with variables $x_{i\oplus 2^j}$ in a constraint, $0 \leq i < 64$, $0 \leq j < 6$. There are 6 such binary constraints for each variable leading to a total of 192 constraints for **S-4**. (Also, refer Subsection 3.2.5.) Table 6.1 demonstrates these differences for each variable. For example, the first entry of the top left table contains x_0 on the left hand column headed by "Row 0 Variable" and six variables $x_1, x_2, x_4, x_8, x_{16}, x_{32}$ on the right-hand-column headed by "Differs from each of the following Variables". The interpretation of this line, using Equation 3.2, leads to the following six constraints for variable x_0:

$$\begin{aligned}
wt(x_0 \oplus x_1) &\geq 2 \\
wt(x_0 \oplus x_2) &\geq 2 \\
wt(x_0 \oplus x_4) &\geq 2 \\
wt(x_0 \oplus x_8) &\geq 2 \\
wt(x_0 \oplus x_{16}) &\geq 2 \\
wt(x_0 \oplus x_{32}) &\geq 2 \qquad (6.5)
\end{aligned}$$

For example, to satisfy criterion **S-4**, variable x_0 in Row 0 should differ from the six variables $x_1, x_2, x_4, x_8, x_{16}$ and x_{32} by at least two bits since S-box input 0 differs from each of 1, 2, 4, 8, 16 and 32 by exactly one bit. We are considering, for row symmetry, only the first of these six variables that are shown in boldface in Table 6.1. Thus the set of variables in each column headed by "Row" in this table and the first of the list of six variables on its right, in the column headed by "Differs from each of the following variables", reside in consecutive rows and are interchangeable, resulting in row symmetry. Table 6.1 should be compared with Figure 6.1 which picturizes the row symmetry, the variables in the odd rows being shown in boldface.

The row symmetry is not conditional. For example, whether the other five variables x_2, x_4, x_8, x_{16} and x_{32} among the six (upon including x_1) participate with x_0 in the symmetry or not, the row symmetry is preserved only because of the first variable x_1 in the set of each of these six variables involved in **S-4**.

6.4.2 Simultaneous Row Interchanges

The reason for the interchanges between rows to be simultaneous is the following. Each constraint listed in a row contains exactly one variable from the third and each listed for the second row contains exactly one from the fourth. For example, consider the six constraints involving x_0. Among these, Equation 6.5 contains the only variable x_{32} from Row 2. The same is true for the other constraints involving the remaining variables.

Interchanging only the first two rows and keeping the third and fourth rows as they are, result in the variables participating in those constraints get interchanged except for the variables of the untouched rows, giving rise to new constraints that are not in **S-4**. Due to the upper bound on the total number of constraints for **S-4**, the new constraints end up *replacing* some of the existing constraints in **S-4**, and are therefore invalid. For example, interchanging Row 0 and Row 1 of Figure 6.1 results in the following constraints for variable x_1:

Row 0 Variable	**Differs from each of the following Variables**	**Row 1 Variable**	**Differs from each of the following Variables**
x_0	$\mathbf{x_1}, x_2, x_4, x_8, x_{16}, x_{32},$	x_1	$\mathbf{x_0}, x_3, x_5, x_9, x_{17}, x_{33},$
x_2	$\mathbf{x_3}, x_0, x_6, x_{10}, x_{18}, x_{34},$	x_3	$\mathbf{x_2}, x_1, x_7, x_{11}, x_{19}, x_{35},$
x_4	$\mathbf{x_5}, x_6, x_0, x_{12}, x_{20}, x_{36},$	x_5	$\mathbf{x_4}, x_7, x_1, x_{13}, x_{21}, x_{37},$
x_6	$\mathbf{x_7}, x_4, x_2, x_{14}, x_{22}, x_{38},$	x_7	$\mathbf{x_6}, x_5, x_3, x_{15}, x_{23}, x_{39},$
x_8	$\mathbf{x_9}, x_{10}, x_{12}, x_0, x_{24}, x_{40},$	x_9	$\mathbf{x_8}, x_{11}, x_{13}, x_1, x_{25}, x_{41},$
x_{10}	$\mathbf{x_{11}}, x_8, x_{14}, x_2, x_{26}, x_{42},$	x_{11}	$\mathbf{x_{10}}, x_9, x_{15}, x_3, x_{27}, x_{43},$
x_{12}	$\mathbf{x_{13}}, x_{14}, x_8, x_4, x_{28}, x_{44},$	x_{13}	$\mathbf{x_{12}}, x_{15}, x_9, x_5, x_{29}, x_{45},$
x_{14}	$\mathbf{x_{15}}, x_{12}, x_{10}, x_6, x_{30}, x_{46},$	x_{15}	$\mathbf{x_{14}}, x_{13}, x_{11}, x_7, x_{31}, x_{47},$
x_{16}	$\mathbf{x_{17}}, x_{18}, x_{20}, x_{24}, x_0, x_{48},$	x_{17}	$\mathbf{x_{16}}, x_{19}, x_{21}, x_{25}, x_1, x_{49},$
x_{18}	$\mathbf{x_{19}}, x_{16}, x_{22}, x_{26}, x_2, x_{50},$	x_{19}	$\mathbf{x_{18}}, x_{17}, x_{23}, x_{27}, x_3, x_{51},$
x_{20}	$\mathbf{x_{21}}, x_{22}, x_{16}, x_{28}, x_4, x_{52},$	x_{21}	$\mathbf{x_{20}}, x_{23}, x_{17}, x_{29}, x_5, x_{53},$
x_{22}	$\mathbf{x_{23}}, x_{20}, x_{18}, x_{30}, x_6, x_{54},$	x_{23}	$\mathbf{x_{22}}, x_{21}, x_{19}, x_{31}, x_7, x_{55},$
x_{24}	$\mathbf{x_{25}}, x_{26}, x_{28}, x_{16}, x_8, x_{56},$	x_{25}	$\mathbf{x_{24}}, x_{27}, x_{29}, x_{17}, x_9, x_{57},$
x_{26}	$\mathbf{x_{27}}, x_{24}, x_{30}, x_{18}, x_{10}, x_{58},$	x_{27}	$\mathbf{x_{26}}, x_{25}, x_{31}, x_{19}, x_{11}, x_{59},$
x_{28}	$\mathbf{x_{29}}, x_{30}, x_{24}, x_{20}, x_{12}, x_{60},$	x_{29}	$\mathbf{x_{28}}, x_{31}, x_{25}, x_{21}, x_{13}, x_{61},$
x_{30}	$\mathbf{x_{31}}, x_{28}, x_{26}, x_{22}, x_{14}, x_{62},$	x_{31}	$\mathbf{x_{30}}, x_{29}, x_{27}, x_{23}, x_{15}, x_{63},$
Row 2 Variable	**Differs from each of the following Variables**	**Row 3 Variable**	**Differs from each of the following Variables**
x_{32}	$\mathbf{x_{33}}, x_{34}, x_{36}, x_{40}, x_{48}, x_0$	x_{33}	$\mathbf{x_{32}}, x_{35}, x_{37}, x_{41}, x_{49}, x_1$
x_{34}	$\mathbf{x_{35}}, x_{32}, x_{38}, x_{42}, x_{50}, x_2$	x_{35}	$\mathbf{x_{34}}, x_{33}, x_{39}, x_{43}, x_{51}, x_3$
x_{36}	$\mathbf{x_{37}}, x_{38}, x_{32}, x_{44}, x_{52}, x_4$	x_{37}	$\mathbf{x_{36}}, x_{39}, x_{33}, x_{45}, x_{53}, x_5$
x_{38}	$\mathbf{x_{39}}, x_{36}, x_{34}, x_{46}, x_{54}, x_6$	x_{39}	$\mathbf{x_{38}}, x_{37}, x_{35}, x_{47}, x_{55}, x_7$
x_{40}	$\mathbf{x_{41}}, x_{42}, x_{44}, x_{32}, x_{56}, x_8$	x_{41}	$\mathbf{x_{40}}, x_{43}, x_{45}, x_{33}, x_{57}, x_9$
x_{42}	$\mathbf{x_{43}}, x_{40}, x_{46}, x_{34}, x_{58}, x_{10}$	x_{43}	$\mathbf{x_{42}}, x_{41}, x_{47}, x_{35}, x_{59}, x_{11}$
x_{44}	$\mathbf{x_{45}}, x_{46}, x_{40}, x_{36}, x_{60}, x_{12}$	x_{45}	$\mathbf{x_{44}}, x_{47}, x_{41}, x_{37}, x_{61}, x_{13}$
x_{46}	$\mathbf{x_{47}}, x_{44}, x_{42}, x_{38}, x_{62}, x_{14}$	x_{47}	$\mathbf{x_{46}}, x_{45}, x_{43}, x_{39}, x_{63}, x_{15}$
x_{48}	$\mathbf{x_{49}}, x_{50}, x_{52}, x_{56}, x_{32}, x_{16}$	x_{49}	$\mathbf{x_{48}}, x_{51}, x_{53}, x_{57}, x_{33}, x_{17}$
x_{50}	$\mathbf{x_{51}}, x_{48}, x_{54}, x_{58}, x_{34}, x_{18}$	x_{51}	$\mathbf{x_{50}}, x_{49}, x_{55}, x_{59}, x_{35}, x_{19}$
x_{52}	$\mathbf{x_{53}}, x_{54}, x_{48}, x_{60}, x_{36}, x_{20}$	x_{53}	$\mathbf{x_{52}}, x_{55}, x_{49}, x_{61}, x_{37}, x_{21}$
x_{54}	$\mathbf{x_{55}}, x_{52}, x_{50}, x_{62}, x_{38}, x_{22}$	x_{55}	$\mathbf{x_{54}}, x_{53}, x_{51}, x_{63}, x_{39}, x_{23}$
x_{56}	$\mathbf{x_{57}}, x_{58}, x_{60}, x_{48}, x_{40}, x_{24}$	x_{57}	$\mathbf{x_{56}}, x_{59}, x_{61}, x_{49}, x_{41}, x_{25}$
x_{58}	$\mathbf{x_{59}}, x_{56}, x_{62}, x_{50}, x_{42}, x_{26}$	x_{59}	$\mathbf{x_{58}}, x_{57}, x_{63}, x_{51}, x_{43}, x_{27}$
x_{60}	$\mathbf{x_{61}}, x_{62}, x_{56}, x_{52}, x_{44}, x_{28}$	x_{61}	$\mathbf{x_{60}}, x_{63}, x_{57}, x_{53}, x_{45}, x_{29}$
x_{62}	$\mathbf{x_{63}}, x_{60}, x_{58}, x_{54}, x_{46}, x_{30}$	x_{63}	$\mathbf{x_{62}}, x_{61}, x_{59}, x_{55}, x_{47}, x_{31}$

Table 6.1: Relationships between the variables participating in Criterion **S-4**

S_8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

(a)

Rows 0 and 1 interchanged, and Rows 2 and 3 interchanged

1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8

(b)

Figure 6.2: (a) The DES S-box S_8. (b) An S-box derived S_8 due to row symmetry, having a score equal to 12

$$\begin{aligned}
wt(x_1 \oplus x_0) &\geq 2 \\
wt(x_1 \oplus x_3) &\geq 2 \\
wt(x_1 \oplus x_5) &\geq 2 \\
wt(x_1 \oplus x_9) &\geq 2 \\
wt(x_1 \oplus x_{17}) &\geq 2 \\
wt(x_1 \oplus x_{32}) &\geq 2 \qquad (6.6)
\end{aligned}$$

Since x_1 is involved in six constraints, criterion **S-4** leads to 192 constraints. In equation 6.6, the 6-bit inputs 1 ($= 000001_2$) and 32 ($= 100000_2$) of the last inequality do not differ by one bit, but by two bits. Criterion **S-4** does not specify what happens in this case. The issue we are having is due to the maximum bound on the total number of constraints, resulting in some valid constraints getting replaced by new constraints such as the last inequality, violating **S-4**.

6.4.3 An Example of Row Symmetry

Example 6.5 *Consider the DES S-box S_8 depicted in Figure 6.2(a). Upon interchanging Row 0 and Row 1, and simultaneously interchanging Row 2 and Row 3, the resulting configuration is depicted in Figure 6.2(b).*

Pairs of variables participating in Equation 6.7				**Columns of Figure 6.3 that are interchangeable**
$(x_0 : x_{12})$	$(x_1 : x_{13})$	$(x_{32} : x_{44})$	$(x_{33} : x_{45})$	Column 0 with Column 6
$(x_2 : x_{14})$	$(x_3 : x_{15})$	$(x_{34} : x_{46})$	$(x_{35} : x_{47})$	Column 1 with Column 7
$(x_4 : x_8)$	$(x_5 : x_9)$	$(x_{36} : x_{40})$	$(x_{37} : x_{41})$	Column 2 with Column 4
$(x_6 : x_{10})$	$(x_7 : x_{11})$	$(x_{38} : x_{42})$	$(x_{39} : x_{43})$	Column 3 with Column 5
$(x_{16} : x_{28})$	$(x_{17} : x_{29})$	$(x_{48} : x_{60})$	$(x_{49} : x_{61})$	Column 8 with Column 14
$(x_{18} : x_{30})$	$(x_{19} : x_{31})$	$(x_{50} : x_{62})$	$(x_{51} : x_{63})$	Column 9 with Column 15
$(x_{20} : x_{24})$	$(x_{21} : x_{25})$	$(x_{52} : x_{56})$	$(x_{53} : x_{57})$	Column 10 with Column 12
$(x_{22} : x_{26})$	$(x_{23} : x_{27})$	$(x_{54} : x_{58})$	$(x_{55} : x_{59})$	Column 11 with Column 13

Table 6.2: Relationships between the variables participating in Criterion **S-5**

6.5 Column Symmetry

A 6×4 S-box exhibits column symmetry if, upon interchanging some or all of its columns, the resulting configuration still satisfies the specified criterion.

Property 6.6 *For a 6×4 S-box, Criterion* **S-5** *exhibits column symmetry but only if the following column interchanges occur simultaneously: Columns 0 and 6, 1 and 7, 2 and 4, 3 and 5, 8 and 14, 9 and 15, 10 and 12, and 11 with 13.*

Proof Let us rewrite Equation 3.3 for a 6×4 S-box, with the decimal number 12 replacing the equivalent binary number 001100_2:

$$\begin{aligned}(\forall i)(\forall j)(0 \leq i, j \leq 63) \wedge (i \neq j) \wedge (i \oplus j &= 12) \\ &\Rightarrow wt(x_i \oplus x_j) \geq 2 \end{aligned} \tag{6.7}$$

Since

$$i \oplus j = 12$$

we have

$$\begin{aligned} i \oplus 12 &= i \oplus (i \oplus j), \text{ from Eq. 6.8} && (6.8)\\ &= j, \text{ by Associativity property of exclusive-OR} && (6.9)\end{aligned}$$

Substituting for j from equation 6.9, equation 6.7 can be rewritten as:

$$(\forall i)(0 \leq i < 32) \wedge wt(x_i \oplus x_{i \oplus 12}) \geq 2 \tag{6.10}$$

Due to the commutativity property of the exclusive-OR operator, equation 6.10 can be rewritten in the following way:

$$(\forall i)(0 \le i < 32) \wedge wt(x_{i \oplus 12} \oplus x_i) \ge 2 \tag{6.11}$$

In other words, variables x_j and $x_{i\oplus 12}$ can be interchanged. For example, for $i = 0, 1, 32, 33$, variables $x_i = x_0, x_1, x_{32}, x_{33}$ reside in Column 0 and the corresponding variables $x_{i\oplus 12} = x_{12}, x_{13}, x_{44}, x_{45}$ reside in Column 6, as shown in figure 6.3, and Columns 0 and 6 can be interchanged. The same reasoning holds for each of the other columns for the other values of i. This completes the proof.

Q.E.D.

Table 6.2 depicts the relationships between the variables participating in Equation 6.7. In the left-hand column, the notation $(x_i : x_j)$ means that there exists a constraint between variables x_i and x_j governed by Equation 6.7. Noting the commutativity of the exclusive-OR operator, a constraint of the form $x_i \oplus x_j$ can be rewritten as $x_j \oplus x_i$, leading to an interchange of these variables. Accordingly, the interpretation in terms of column interchanges is given in the right-hand column of Table 6.2. Figure 6.3 additionally shows the interchangeable columns identically formatted. An observation visible in Table 6.2 is that interchanges within columns 0 to 7 are identical with those within columns 8 to 15. For this reason and to add clarity, Figure 6.3 shows the only the first eight interchangeable columns.

	$x_1x_2x_3x_4$										
x_0x_5	0	1	2	3	4	5	6	7	8	$\dots$	15
0	$\mathbf{x_0}$	$[x_2]$	$\overline{x_4}$	(x_6)	$\overline{x_8}$	(x_{10})	$\mathbf{x_{12}}$	$[x_{14}]$	x_{16}	$\dots$	x_{30}
1	$\mathbf{x_1}$	$[x_3]$	$\overline{x_5}$	(x_7)	$\overline{x_9}$	(x_{11})	$\mathbf{x_{13}}$	$[x_{15}]$	x_{17}	$\dots$	x_{31}
2	$\mathbf{x_{32}}$	$[x_{34}]$	$\overline{x_{36}}$	(x_{38})	$\overline{x_{40}}$	(x_{42})	$\mathbf{x_{44}}$	$[x_{46}]$	x_{48}	$\dots$	x_{62}
3	$\mathbf{x_{33}}$	$[x_{35}]$	$\overline{x_{37}}$	(x_{39})	$\overline{x_{41}}$	(x_{43})	$\mathbf{x_{45}}$	$[x_{47}]$	x_{49}	$\dots$	x_{63}

Figure 6.3: **S-5** for a 6×4 S-box exhibiting Column Symmetry. Interchangeable columns are formatted identically. Only columns $0 - 7$ are shown, and columns $8 - 15$ are similarly interchangeable.

Example 6.6 *Consider the DES S-box S_8 depicted in Figure 6.4(a). Upon simultaneously interchanging columns 0 and 6, 1 and 7, 2 and 4, 3 and 5, 8 and 14, 9 and 15, 10 and 12, and 11 with 13, the resulting configuration is depicted in Figure 6.4(b). This configuration is an S-box with a score equal to 12.*

6.5.1 Simultaneous Column Interchanges

All eight pairs of columns should be interchanged simultaneously for the resulting configuration to be an S-box satisfying all criteria. Let us see what happens if, for

S_8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

(a)

After simultaneous column interchanges

11	1	6	15	8	4	13	2	12	7	5	0	3	14	10	9
7	4	10	3	13	8	1	15	9	2	0	14	6	11	12	5
14	2	9	12	4	1	7	11	5	8	15	3	10	13	0	6
8	13	4	10	14	7	2	1	6	11	3	5	9	0	15	12

(b)

Figure 6.4: (*a*) The DES S-box S_8. (*b*) An S-box derived from S_8 due to column symmetry having a score equal to 12

example, exactly one pair of columns, say, column 0 and column 6, is interchanged, with the remaining seven pairs of columns kept intact. We want to see the effect of this interchange on one of the other criteria, say, **S-4**. This interchange entails swapping of the variables in the following pairs: (x_0, x_{12}), (x_1, x_{13}), (x_{32}, x_{44}) and (x_{33}, x_{45}). Let us rewrite a portion of Table 6.2 that involves only these variables in the left-hand-column, before and after column interchange. The result is Table 6.3.

Similar to the problem in row symmetry when row interchanges are not simultaneous, new constraints now arise that are not in **S-4**. Due to the upper bound on the total number of constraints for **S-4**, the new constraints end up *replacing* some of the existing constraints resulting in **S-4** getting violated. In this example, the following *new* constraints for variable x_2 are derived from Table 6.3.

$$\begin{aligned} wt(x_{12} \oplus x_2) &\geq 2 \\ wt(x_{16} \oplus x_2) &\geq 2 \end{aligned} \tag{6.12}$$

As we have seen earlier, criterion **S-4** leads to 192 constraints. In equation 6.12, the 6-bit inputs 12 ($= 001100_2$) and 2 ($= 000010_2$) of the first inequality do not differ by one bit, but by three bits. The same is true of the second inequality, for inputs 16 and 2. Criterion **S-4** does not specify what happens in this case. The issue we are having is that due to the maximum bound on the total number of constraints for **S-4**, some valid constraints are replaced by these new inequalities, violating **S-4**.

Row 0 Variable	**Differs from each of the following Variables**	**Row 1 Variable**	**Differs from each of the following Variables**
x_0	$\mathbf{x_1}, x_2, x_4, x_8, x_{16}, x_{32},$	x_1	$\mathbf{x_0}, x_3, x_5, x_9, x_{17}, x_{33}$
x_{12}	$\mathbf{x_{13}}, x_{14}, x_8, x_4, x_{28}, x_{44},$	x_{13}	$\mathbf{x_{12}}, x_{15}, x_9, x_5, x_{29}, x_{45}$
Row 2 Variable	**Differs from each of the following Variables**	**Row 3 Variable**	**Differs from each of the following Variables**
x_{32}	$\mathbf{x_{33}},\ x_{34},\ x_{36},\ x_{40},\ x_{48},\ x_0$	x_{33}	$\mathbf{x_{32}},\ x_{35},\ x_{37},\ x_{41},\ x_{49},\ x_1$
x_{44}	$\mathbf{x_{45}},\ x_{46},\ x_{40},\ x_{36},\ x_{60},\ x_{12}$	x_{45}	$\mathbf{x_{44}},\ x_{47},\ x_{41},\ x_{37},\ x_{61},\ x_{13}$

(a) Before interchanging columns 0 and 6

Row 0 Variable	**Differs from each of the following Variables**	**Row 1 Variable**	**Differs from each of the following Variables**
x_{12}	$\mathbf{x_{13}}, x_2, x_4, x_8, x_{16}, x_{32},$	x_{13}	$\mathbf{x_{12}}, x_3, x_5, x_9, x_{17}, x_{33}$
x_0	$\mathbf{x_1}, x_{14}, x_8, x_4, x_{28}, x_{44},$	x_1	$\mathbf{x_0}, x_{15}, x_9, x_5, x_{29}, x_{45}$
Row 2 Variable	**Differs from each of the following Variables**	**Row 3 Variable**	**Differs from each of the following Variables**
x_{44}	$\mathbf{x_{45}},\ x_{34},\ x_{36},\ x_{40},\ x_{48},\ x_0$	x_{45}	$\mathbf{x_{44}},\ x_{35},\ x_{37},\ x_{41},\ x_{49},\ x_1$
x_{32}	$\mathbf{x_{33}},\ x_{46},\ x_{40},\ x_{36},\ x_{60},\ x_{12}$	x_{33}	$\mathbf{x_{32}},\ x_{47},\ x_{41},\ x_{37},\ x_{61},\ x_{13}$

(b) After interchanging columns 0 and 6

Table 6.3: Relationships between the variables participating in Criterion **S-4** before and after interchanging columns 0 and 6. Not all relationships are shown.

6.6 Diagonal Symmetry

A 6×4 S-box exhibits diagonal symmetry if, upon interchanging some or all of its elements in a diagonal-wise fashion, the resulting configuration still satisfies the specified criterion. For our purposes, imagine the S-box to be divided into four equal-sized rectangular quadrants. Figure 6.5 illustrates the idea.

Property 6.7 *For a 6×4 S-box, Criterion* **S-6** *exhibits diagonal symmetry but only if the following diagonal interchanges occur simultaneously:*

1. *The top left and bottom right quadrants, and*

2. *The top right and bottom left quadrants*

S_8	$x_1x_2x_3x_4$															
x_0x_5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	x_0	x_2	x_4	x_6	x_8	x_{10}	x_{12}	x_{14}	$\mathbf{x_{16}}$	$\mathbf{x_{18}}$	$\mathbf{x_{20}}$	$\mathbf{x_{22}}$	$\mathbf{x_{24}}$	$\mathbf{x_{26}}$	$\mathbf{x_{28}}$	$\mathbf{x_{30}}$
1	x_1	x_3	x_5	x_7	x_9	x_{11}	x_{13}	x_{15}	$\mathbf{x_{17}}$	$\mathbf{x_{19}}$	$\mathbf{x_{21}}$	$\mathbf{x_{23}}$	$\mathbf{x_{25}}$	$\mathbf{x_{27}}$	$\mathbf{x_{29}}$	$\mathbf{x_{31}}$
2	$\mathbf{x_{32}}$	$\mathbf{x_{34}}$	$\mathbf{x_{36}}$	$\mathbf{x_{38}}$	$\mathbf{x_{40}}$	$\mathbf{x_{42}}$	$\mathbf{x_{44}}$	$\mathbf{x_{46}}$	x_{48}	x_{50}	x_{52}	x_{54}	x_{56}	x_{58}	x_{60}	x_{62}
3	$\mathbf{x_{33}}$	$\mathbf{x_{35}}$	$\mathbf{x_{37}}$	$\mathbf{x_{39}}$	$\mathbf{x_{41}}$	$\mathbf{x_{43}}$	$\mathbf{x_{45}}$	$\mathbf{x_{47}}$	x_{49}	x_{51}	x_{53}	x_{55}	x_{57}	x_{59}	x_{61}	x_{63}

Figure 6.5: **S-6** for a 6×4 S-box exhibiting Diagonal Symmetry. Interchangeable quadrants have entries formatted identically.

Proof Let us rewrite Equation 3.4 for a 6×4 S-box, with the decimal numbers 51 and 48 replacing the equivalent binary number 110011_2 and 110000_2:

$$\begin{aligned}(\forall i)(\forall j)(0 \le i < j \le 63), [(i \oplus j) \wedge 51] &= 48 \\ &\Rightarrow x_i \neq x_j \end{aligned} \tag{6.13}$$

It is easy to check that if $i = 0$, then $j = 48, 52, 56, 60$. If $i = 1$, then $j = 49, 53, 57, 61$, and so on. The consequent inequality-relationships between x_i and each of the four variables x_j participating in the constraint 6.13 are depicted in Table 6.4. Note that since x_0 is not equal to each of x_{48}, x_{52}, x_{56} and x_{60}, the reverse is also true, namely, that x_0 will appear as one of the four variables in each row for x_{48}, x_{52}, x_{56} and x_{60}, and likewise for all variables constituting the S-box. Accordingly, the total number of constraints that Equation 6.13 gives rise to is 128, as discussed in subsection 3.2.7.

By arranging variables $x_0, x_1, \ldots, x_{15}$ their related quadruplets arrange to $x_{48}, x_{49}, \ldots, x_{63}$. These two form the top-left and bottom-right quadrants of the S-box respectively, as illustrated in Figure 6.5. The same is true with the other two sets of variables, namely, $x_{16}, x_{17}, \ldots, x_{33}$ and their related quadruplets $x_{32}, x_{33}, \ldots, x_{47}$, that respectively form the top-right and bottom-left quadrants of the S-box of Figure 6.5. Due to the commutativity property of the exclusive-OR operator, the rectangles are diagonally interchangeable. This completes the proof. **Q.E.D.**

6.6.1 Simultaneous Interchanges of Diagonal Rectangles

The quadrants should be diagonally interchanged simultaneously. Let us see why. It is possible for two variables not participating in any constraint other than S-6, to have equal values. Assume that $x_0 = x_{46}$. If we interchange the bottom-left and top-right quadrants *without interchanging the other two*, x_{46} will appear on

Variable	**Differs from each of the following variables**	**Variable**	**Differs from each of the following variables**
x_0	x_{48}, x_{52} x_{56} x_{60}	x_1	x_{49}, x_{53} x_{57} x_{61}
x_2	x_{50}, x_{54} x_{58} x_{62}	x_3	x_{51}, x_{55} x_{59} x_{63}
x_4	x_{52}, x_{48} x_{60} x_{56}	x_5	x_{53}, x_{49} x_{61} x_{57}
x_6	x_{54}, x_{50} x_{62} x_{58}	x_7	x_{55}, x_{51} x_{63} x_{59}
x_8	x_{56}, x_{60} x_{48} x_{52}	x_9	x_{57}, x_{61} x_{49} x_{53}
x_{10}	x_{58}, x_{62} x_{50} x_{54}	x_{11}	x_{59}, x_{63} x_{51} x_{55}
x_{12}	x_{60}, x_{56} x_{52} x_{48}	x_{13}	x_{61}, x_{57} x_{53} x_{49}
x_{14}	x_{62}, x_{58} x_{54} x_{50}	x_{15}	x_{63}, x_{59} x_{55} x_{51}
x_{16}	x_{32}, x_{36} x_{40} x_{44}	x_{17}	x_{33}, x_{37} x_{41} x_{45}
x_{18}	x_{34}, x_{38} x_{42} x_{46}	x_{19}	x_{35}, x_{39} x_{43} x_{47}
x_{20}	x_{36}, x_{32} x_{44} x_{40}	x_{21}	x_{37}, x_{33} x_{45} x_{41}
x_{22}	x_{38}, x_{34} x_{46} x_{42}	x_{23}	x_{39}, x_{35} x_{47} x_{43}
x_{24}	x_{40}, x_{44} x_{32} x_{36}	x_{25}	x_{41}, x_{45} x_{33} x_{37}
x_{26}	x_{42}, x_{46} x_{34} x_{38}	x_{27}	x_{43}, x_{47} x_{35} x_{39}
x_{28}	x_{44}, x_{40} x_{36} x_{32}	x_{29}	x_{45}, x_{41} x_{37} x_{33}
x_{30}	x_{46}, x_{42} x_{38} x_{34}	x_{31}	x_{47}, x_{43} x_{39} x_{35}
x_{32}	x_{16}, x_{20} x_{24} x_{28}	x_{33}	x_{17}, x_{21} x_{25} x_{29}
x_{34}	x_{18}, x_{22} x_{26} x_{30}	x_{35}	x_{19}, x_{23} x_{27} x_{31}
x_{36}	x_{20}, x_{16} x_{28} x_{24}	x_{37}	x_{21}, x_{17} x_{29} x_{25}
x_{38}	x_{22}, x_{18} x_{30} x_{26}	x_{39}	x_{23}, x_{19} x_{31} x_{27}
x_{40}	x_{24}, x_{28} x_{16} x_{20}	x_{41}	x_{25}, x_{29} x_{17} x_{21}
x_{42}	x_{26}, x_{30} x_{18} x_{22}	x_{43}	x_{27}, x_{31} x_{19} x_{23}
x_{44}	x_{28}, x_{24} x_{20} x_{16}	x_{45}	x_{29}, x_{25} x_{21} x_{17}
x_{46}	x_{30}, x_{26} x_{22} x_{18}	x_{47}	x_{31}, x_{27} x_{23} x_{19}
x_{48}	x_0, x_4 x_8 x_{12}	x_{49}	x_1, x_5 x_9 x_{13}
x_{50}	x_2, x_6 x_{10} x_{14}	x_{51}	x_3, x_7 x_{11} x_{15}
x_{52}	x_4, x_0 x_{12} x_8	x_{53}	x_5, x_1 x_{13} x_9
x_{54}	x_6, x_2 x_{14} x_{10}	x_{55}	x_7, x_3 x_{15} x_{11}
x_{56}	x_8, x_{12} x_0 x_4	x_{57}	x_9, x_{13} x_1 x_5
x_{58}	x_{10}, x_{14} x_2 x_6	x_{59}	x_{11}, x_{15} x_3 x_7
x_{60}	x_{12}, x_8 x_4 x_0	x_{61}	x_{13}, x_9 x_5 x_1
x_{62}	x_{14}, x_{10} x_6 x_2	x_{63}	x_{15}, x_{11} x_7 x_3

Table 6.4: Relationships between the variables participating in Criterion **S-6**

the top row. But this violates criterion **S-3** that states that each row should be a

S_8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

(a)

Quadrants diagonally interchanged

0	6	10	13	15	3	5	8	7	11	4	1	9	12	14	2
15	12	9	0	3	5	6	11	2	1	14	7	4	10	8	13
10	9	3	14	5	0	12	7	13	2	8	4	6	15	11	1
12	5	6	11	0	14	9	2	1	15	13	8	10	3	7	4

(b)

Figure 6.6: (*a*) The DES S-box S_8 split into four quadrants. (*b*) A configuration obtained due to Diagonal Symmetry, having a score equal to 12.

one-one permutation of $\mathbb{Z}_{16}$. To satisfy **S-3**, the other two quadrants should also be interchanged.

Example 6.7 *Consider the DES S-box* **S-8** *depicted in Figure 6.6(a) as split into four quadrants. Upon interchanging the top-left and bottom-right quadrants, and simultaneously interchanging the other two quadrants, the resulting configuration is depicted in Figure 6.6(b). This configuration is an S-box that satisfies the other criteria and has a score equal to 12.*

6.7 Is the Resulting Configuration Always an S-box?

When an S-box is transformed to another configuration using either or row, column, and diagonal symmetry property of the appropriate constraint that models the particular criterion, only the particular constraint is satisfied. We have seen that the transformed configuration is not necessarily an S-box because other constraint(s) may get violated due to the transformation. For the transformed configuration to be an S-box, the interchanges have to be simultaneous in each type of symmetry.

The other problem is that the score of the transformed configuration *may* exceed the specified threshold, violating criterion **S-2** specified by constraint 4.7. It turns out that the S-box of Figure 6.1(b) is an alternate S-box having a score of 12 and satisfying all of the constraints. This score is the same as that of the

S_8	$x_1x_2x_3x_4$															
x_0x_5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	x_{63}	x_{61}	x_{59}	x_{57}	x_{55}	x_{53}	x_{51}	x_{49}	x_{47}	x_{45}	x_{43}	x_{41}	x_{39}	x_{37}	x_{35}	x_{33}
1	x_{62}	x_{60}	x_{58}	x_{56}	x_{54}	x_{52}	x_{50}	x_{48}	x_{46}	x_{44}	x_{42}	x_{40}	x_{38}	x_{36}	x_{34}	x_{32}
2	x_{31}	x_{29}	x_{27}	x_{25}	x_{23}	x_{21}	x_{19}	x_{17}	x_{15}	x_{13}	x_{11}	x_9	x_7	x_5	x_3	x_1
3	x_{30}	x_{28}	x_{26}	x_{24}	x_{22}	x_{20}	x_{18}	x_{16}	x_{14}	x_{12}	x_{10}	x_8	x_6	x_4	x_2	x_0

Figure 6.7: **S-7** for a 6×4 S-box exhibiting Rotational Symmetry

original DES S-box **S-8**, i.e., has not changed for this example. The same is true of the configuration of Figure 6.5, that it is an S-box with score 12.

In the next two sections, we discuss Rotational Symmetry and Bit Inversion Symmetry. We prove that these two forms of symmetry always yield an S-box upon appropriate transformation, and that their scores will never change.

6.8 Rotational Symmetry

An S-box exhibits rotational symmetry with respect to a constraint if, upon rotating the same by 180° about both, its top and left edges, the resulting configuration still satisfies the constraint.

Property 6.8 *Criterion* **S-7** *exhibits rotational symmetry for a* 6×4 *S-box.*

Proof Let us rewrite the COUNT constraint of criterion **S-7** given by Equation 5.1 for a 6×4 S-box:

$$\bigwedge_{i=0}^{31} f(x_i \oplus x_{63-i}) \leq 8 \tag{6.14}$$

By the commutativity property of the exclusive-OR operation, x_i and x_{63-i} are interchangeable. Interchanging the values of these variables therefore does not affect Equation 6.14. Repositioning the S-box entries results in a configuration obtained by rotating the S-box to an upside-down position (Figure 6.7). This proves the proposition.

Q.E.D.

6.8.1 *Impact on Constraints for Criteria* **S-2** *to* **S-6**

Upon applying a 180°–rotation about its top and left edges, as Figure 6.7 suggests, each row still has unique entries suggesting that criterion **S-3** is unaffected. All of the criteria **S-4**, **S-5** and **S-6** are similarly preserved, and are easily verified using Tables 6.1, 6.2 and Table 6.4 should be analyzed. For example, reading the rows of Table 6.4 backwards results in traversing the columns of Figure 6.7 whose column-numbers are specified in the Table.

Let us examine how the score of the configuration of Figure 6.7 is affected, that is, whether S-box rotation has an impact on criterion **S-2**.

Property 6.9 *The score $\sigma_X(\Phi)$ of an S-box Φ is unaffected by S-box rotation.*

Proof Consider two entries $x_i = v$ and $x_{\bar{i}} = w$ of an S-box. We are going to interchange the values to these variables by setting $x_i = v$, studying the layered linear approximation table for this assignment, setting $x_{\bar{i}}$ and analyzing the corresponding layered linear approximation table, and adding the entries in the two tables to examine the effect of both assignments.

If $x_i = v$ is moved to $x_{\bar{i}} = v$, all rows in the layered linear approximation table for $x_{\bar{i}} = v$ numbered by a having odd parity get inverted while those having even parity do not change, due to Property 6.3 . Upon simultaneously assigning $x_i = w$, all rows a' in the layered table for the entry $x_i = w$ get similarly inverted if a' has odd parity, and do not change for even parity. Adding these two tables results in a linear approximation table for $x_i = v$ and $x_{\bar{i}} = w$ that is identical to one for $x_i = w$ and $x_{\bar{i}} = v$. This is now extended to all entries in the S-box, resulting in its final, cumulative linear approximation table not changing when all of its bits are inverted. As such, the score does not change.

Q.E.D.

Example 6.8 *Consider the DES S-box S_8 depicted in Figure 6.8(a). Upon inverting this S-box by $180°$, Figure 6.8(b) depicts the resulting S-box having a score equal to 12, the same as that of S_8.*

6.9 Bit Inversion Symmetry

An S-box exhibits inversion symmetry if, upon replacing all of its entries by their one's-complements, the resulting configuration is still an S-box. As we have seen thus far, the constraint for criteria **S-4** exhibits row symmetry, that for **S-5** possesses column symmetry, **S-6** has diagonal and **S-7**, rotational symmetry. We would like to know which particular constraint possess bit inversion symmetry.

Property 6.10 *All constraints modeling criteria* **S-3** *to* **S-7** *possess bit inversion symmetry.*

Proof The result follows at once from the exclusive-OR property of invariance to complementation of its two operands: $A \oplus B = \overline{A} \oplus \overline{B}$ for two n-bit quantities A and B. It also follows from the fact that $A \neq B$ is equivalent to $\overline{A} \neq \overline{B}$. For, $A \neq B \Rightarrow A \oplus B \neq 0 \Rightarrow \overline{A} \oplus \overline{B} \neq 0 \Rightarrow \overline{A} \neq \overline{B}$.

Q.E.D.

To get a further insight into the proof, the constraints modeling each criteria **S-3** to **S-6** are studied as follows:

1. **S-3**: This is the `Alldiff` constraint on the variables in each row, which remain different upon complementing the entries in that row. Hence **S-3** is unaffected by bit inversion, exhibiting symmetry.

2. **S-4**: For $0 \leq i < 32$,
$$wt(x_{2i} \oplus x_{2i+1}) \geq 2$$
is equivalent to:
$$wt(\overline{x_{2i}} \oplus \overline{x_{2i+1}}) \geq 2$$
suggesting that **S-4** is unaffected by inverting the bits of all S-box entries, exhibiting bit inversion symmetry.

S_8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

(a)

Rotation about the Top and Left Edges by 180°

11	6	5	3	0	9	12	15	13	8	10	4	7	14	1	2
8	5	3	15	13	10	6	0	2	14	12	9	1	4	11	7
2	9 7 14	0	11	6	5	12	4	7	3	10	8	13	15	1	
7	12	0	5	14	3	9	10	1	11	15	6	4	8	2	13

(b)

Figure 6.8: (*a*) The DES S-box S_8. (*b*) The S-box obtained due to Rotational Symmetry, having a score equal to 12.

3. **S-5**: Equation 6.7 is rewritten as

$$\begin{aligned}(\forall i)(\forall j)(0 \leq i, j \leq 63) \wedge (i \neq j) \wedge (i \oplus j &= 12) \\ \Rightarrow wt(x_i \oplus x_j) \geq 2\end{aligned}$$

   This equation is equivalent to the following:

$$\begin{aligned}(\forall i)(\forall j)(0 \leq i, j \leq 63) \wedge (i \neq j) \wedge (i \oplus j &= 12) \\ \Rightarrow wt(\overline{x_i} \oplus \overline{x_j}) \geq 2\end{aligned}$$

   suggesting invariance of **S-5** over bitwise complementation.

4. **S-6**: Equation 6.13 is rewritten as

$$\begin{aligned}(\forall i)(\forall j)(0 \leq i < j \leq 63), [(i \oplus j) \wedge 51] &= 48 \\ \Rightarrow x_i \neq x_j\end{aligned}$$

   This equation is equivalent to:

$$\begin{aligned}(\forall i)(\forall j)(0 \leq i < j \leq 63), [(i \oplus j) \wedge 51] &= 48 \\ \Rightarrow \overline{x_i} \neq \overline{x_j}\end{aligned}$$

   suggesting invariance of **S-6** over bitwise complementation.

5. **S-7**: The COUNT constraint is modeled Equation 6.14, rewritten as follows:

$$\bigwedge_{i=0}^{31} f(x_i \oplus x_{\bar{i}}) \leq 8$$

   This equation is equivalent to:

$$\bigwedge_{i=0}^{31} f(\overline{x_i} \oplus \overline{x_{\bar{i}}}) \leq 8$$

   suggesting invariance of **S-7** over bitwise complementation.

6.9.1 Effect of Bit Inversion on the Score of an S-box

We have seen how bit inversion did not affect the `Alldiff` constraint modeling criterion **S-3**, binary constraints modeling criteria **S-4** to **S-6**, and the COUNT constraint for **S-7**. Will the resulting configuration still remain an S-box? In other words, what happens to its score? The following property establishes this important fact.

Property 6.11 *The score $\sigma_X(\Phi)$ of an S-box Φ does not change upon bitwise complementation of the entries in Φ.*

Proof Consider two entries $x_i = v$ and $x_j = \overline{v}$ of an S-box. We are going to interchange the values to these variables by setting $x_i = \overline{v}$, studying the layered linear approximation table for this assignment, setting $x_j = \overline{v}$ and analyzing the corresponding layered linear approximation table, and adding the entries in the two tables to examine the effect of both assignments.

If $x_i = v$ is changed to $x_i = \overline{v}$, all of its entries in the columns of the layered linear approximation table for $x_i = v$ that are headed by b having odd parity get inverted due to Property 6.4. Upon simultaneously changing to $x_j = v$ from the earlier assignment $x_j = \overline{v}$, the corresponding columns for the entry $x_j = \overline{v}$ get similarly inverted in its layered linear approximation table. Adding these two tables results in a linear approximation table for $x_i = v$ and $x_j = \overline{v}$ that is identical to one for $x_i = \overline{v}$ and $x_j = v$. This is now extended to all entries in the S-box, resulting in its final, cumulative linear approximation table not changing when all of its bits are inverted. As such, the score does not change.

Q.E.D.

Example 6.9 *Consider the DES S-box S_8 depicted in Figure 6.9(a). Upon replacing each entry in this S-box by its one's-complement (by subtracting each entry from $2^4 - 1 = 15$), Figure 6.9(b) depicts the resulting S-box having a score equal to 12, the same as that of S_8.*

6.10 The Multiple S-box Problem

The S-box criterion **S-8** for multiple S-boxes is now discussed along with an example. The criterion for three S-boxes, mentioned in Chapter 3, is repeated here for convenience:

S-8 "Similar to **S-7**, but with stronger restrictions in the case $\Delta O_{i,j} = 0$ for the case of three active S-boxes on round i." [16].

Let us first see what an *active* S-box is.

6.10.1 Active S-Boxes in a particular Round

Given a probable bit pattern, an S-box S_j ($1 \leq j \leq 8$ is said to be *active* on a round i of encryption / decryption if the difference $\Delta I_{i,j}$ between inputs m_i and m'_i to S-box S_j, in round i, are not all zero during this round. Then $\Delta I_{i,j} = m_i \oplus m'_i$ [16]. Now $\Delta I_{i,j} = 0 \Rightarrow m_i = m'_i$, that is, two different messages to be input to the same S-box S_j have identical content. If this is so, then the S-box is inactive. As the

S_8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

(a)

Entries with Bits Complemented

2	13	7	11	9	0	4	14	5	6	12	1	10	15	3	8
14	0	2	7	5	12	8	11	3	10	9	4	15	1	6	13
8	4	11	14	6	3	1	13	15	9	5	2	0	12	10	7
13	14	1	8	11	5	7	2	0	3	6	15	12	10	9	4

(b)

Figure 6.9: (*a*) The DES S-box S_8. (*b*) The S-box obtained due to Bit Inversion Symmetry, having a score equal to 12.

number of rounds i increases, the number of active S-boxes also increases [16]. Criterion **S-8** deals with active S-boxes taken three at a time, each having inputs not always identical when compared pair-wise. The S-boxes are listed as S_j, $S_{j \bmod 8+1}$ and $S_{(j \bmod 8+1) \bmod 8+1}$, $1 \leq j \leq 8$, S_8 and S_1 being treated as adjacent to each other.

Similar to the Linear Approximation Table, an *XOR Distribution Table* [11] is constructed. This table considers differences between two inputs to an S-box and between their corresponding outputs, and is used by Biham and Shamir in the differential cryptanalysis of DES.

6.10.2 XOR Distribution Table

In this table, differences of inputs to an S-box S_j and differences in the outputs of S_j are considered. Consider two inputs k and k' to the same S-box S_j, with corresponding outputs x_k and x'_k. Determine the following two differences:

$$\begin{aligned}\Delta k &= k \oplus k' \\ \Delta x_k &= x_k \oplus x'_k\end{aligned}$$

Each entry in the differential approximation table is defined as follows, for a $n \times m$ S-box S_j.

b	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a																
0	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	6	0	2	4	4	0	10	12	4	10	6	2	4
2	0	0	0	8	0	4	4	4	0	6	8	6	12	6	4	2
3	14	4	2	2	10	6	4	2	6	4	4	0	2	2	2	0
4	0	0	0	6	0	10	10	6	0	4	6	4	2	8	6	2
5	4	8	6	2	2	4	4	2	0	4	4	0	12	2	4	6
6	0	4	2	4	8	2	6	2	8	4	4	2	4	2	0	12
7	2	4	10	4	0	4	8	4	2	4	8	2	2	2	4	4
8	0	0	0	12	0	8	8	4	0	6	2	8	8	2	2	4
9	0	2	4	0	2	4	6	0	2	2	8	0	10	0	2	12
10	0	8	6	2	2	8	6	0	6	4	6	0	4	0	2	10
11	2	4	0	10	2	2	4	0	2	6	2	6	6	4	2	12
12	0	0	0	8	0	6	6	0	0	6	6	4	6	6	14	2
13	6	6	4	8	4	8	2	6	0	6	4	6	0	2	0	2
14	0	4	8	8	6	6	4	0	6	6	4	0	0	4	0	8
15	2	0	2	4	4	6	4	2	4	8	2	2	2	6	8	8
16	0	0	0	0	0	0	2	14	0	6	6	12	4	6	8	6
17	6	8	2	4	6	4	8	6	4	0	6	6	0	4	0	0
18	0	8	4	2	6	6	4	6	6	4	2	6	6	0	4	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
59	2	6	4	0	0	2	4	6	4	6	8	6	4	4	6	2
60	0	10	4	0	12	0	4	2	6	0	4	12	4	4	2	0
61	0	8	6	2	2	6	0	8	4	4	0	4	0	12	4	4
62	4	8	2	2	2	4	4	14	4	2	0	2	0	8	4	4
63	4	8	4	2	4	0	2	4	4	2	4	8	8	6	2	2

Table 6.5: Tabulating the counts $D_1(a, b)$ for the S-box S_1 of Figure 1.1

$$D_j(a, b) = \#\{k : \Delta k = \Delta x_k; \Delta k \in \mathbb{Z}_{2^n}; \Delta x_k \in \mathbb{Z}_{2^m}\} \tag{6.15}$$

Example 6.10 *The differential approximation table of S-box S_1 used in [11], is constructed as shown in Table 6.5. Each entry is denoted as $D_1(a, b)$ for S-box S_1, where a is the row number and b, the column number of the table, $0 \leq a < 64, 0 \leq b < 16$.*

Note the difference between equations 6.15 and 4.2. In equation 4.2, it is the *linear combination* of *subsets* of input and output bits that are considered for equality. The definition of $D^{\Phi}(a,b)$ is simpler. Equation 6.15 records counts of equalities of merely *differences* between any two inputs k and k' ($\Delta k = k \oplus k'$), and their corresponding outputs x_k and $x_{k'}$ ($\Delta x_k = x_k \oplus x_{k'}$) of an S-box.

6.10.3 Probability Measure

The probability of an output difference $\Delta x_k = b$ given that the corresponding input difference $\Delta k = a$, for a 6×4 S-boxS_j is approximated by the following equation:

$$P(\Delta x_k = b | \Delta k = a) = \frac{D_j(a,b)}{64} \tag{6.16}$$

Example 6.11 *For the DES S-box S_1 of figure 1.1, the probability that two outputs differ by 5 given that the corresponding inputs differ by 7 is approximated as follows:*

$$\begin{aligned} P(\Delta x_k = 5 | \Delta k = 7) &= \frac{D_1(7,5)}{64} \\ &= \frac{4}{64}, \textit{ from Row 7, Column 5 of Table 6.5} \\ &= \frac{1}{16} \end{aligned}$$

6.10.4 Attack on an S-box with Highest Probability

The following bit-patterns are derived [16] by taking three adjacent S-boxes $S_j, S_{j \bmod 8+1}$ and $S_{(j \bmod 8+1) \bmod 8+1}$, where $1 \le j \le 8$, in order to simplify the analysis:

$$\begin{aligned} \Delta I_{i,j} &= 00cd11_2, c,d \in \mathbb{Z}_2 \\ \Delta I_{i,j \bmod 8+1} &= 11gh10_2, g,h \in \mathbb{Z}_2 \\ \Delta I_{i,(j \bmod 8+1) \bmod 8+1} &= 10km00_2, k,m \in \mathbb{Z}_2 \end{aligned}$$

The objective is to minimize the highest probability of success of an attack. To find the highest probability of success, determine c, d so as to maximize the conditional probability for one S-box S_j. Similarly, determine g, h for the adjacent S-box $S_{j \bmod 8+1}$, and k, m for the next-adjacent S-box $S_{(j \bmod 8+1) \bmod 8+1}$ to maximize the respective highest probabilities.

For an S-box S_j, $1 \le j \le 8$, let

$$\begin{aligned}
q_{0,j} &= P(\Delta x_r = 0 | \Delta r = 00cd11_2) \\
&= \max_{c,d} \frac{D_j(00cd11_2, 0)}{64} \text{ from Eq. 6.16} \\
&= \frac{\max\{D_j(3,0), D_j(7,0), D_j(11,0), D_j(15,0)\}}{64} \qquad (6.17) \\
q_{1,j} &= \max_{g,h} P(\Delta x_r = 0 | \Delta r = 11gh10_2) \\
&= \max_{g,h} \frac{D_j(11gh10_2, 0)}{64} \text{ from Eq. 6.16} \\
&= \frac{\max\{D_j(50,0), D_j(54,0), D_j(58,0), D_j(62,0)\}}{64} \qquad (6.18) \\
q_{2,j} &= \max_{k,m} P(\Delta x_r = 0 | \Delta r = 10km00_2) \\
&= \max_{k,m} \frac{D_j(10km00_2, 0)}{64} \text{ from Eq. 6.16} \\
&= \frac{\max\{D_j(32,0), D_j(36,0), D_j(40,0), D_j(44,0)\}}{64} \qquad (6.19)
\end{aligned}$$

Upon determining $q_{0,j}$, $q_{1,j \bmod 8+1}$ and $q_{2,(j \bmod 8+1) \bmod 8+1}$ for three adjacent S-boxes, the highest probability of success of an attack (assuming independence) is equal to their product, namely,

$$q_{0,j} \cdot q_{1,j \bmod 8+1} \cdot q_{2,(j \bmod 8+1) \bmod 8+1}$$

.

Example 6.12 *Consider the DES S-box S_1 ($j = 1$) used in Example 6.10. From equation 6.17,*

$$q_{0,1} = \frac{\max\{14, 2, 2, 2\}}{64} = \frac{14}{64} = \frac{7}{32}$$

by looking up column 0 under rows 3, 7, 11 and 15 of the XOR table of S-box S_1, namely Table 6.5, and substituting in equation 6.17. Similarly for S-box S_2 adjacent to S_1, equation 6.18 gives

$$q_{1,2} = \frac{\max\{8, 8, 4, 4\}}{64} = \frac{8}{64} = \frac{1}{8}$$

by looking up column 0 under rows 50, 54, 58 and 62 of its XOR table. For the next adjacent S-box S_3, we similarly look up column 0 under rows 32,36,40,44 of its XOR table and substitute in equation 6.19 to yield

$$q_{2,3} = \frac{\max\{0,4,6,10\}}{64} = \frac{10}{64} = \frac{5}{32}$$

The highest probability of success of an attack is now

$$\begin{aligned} &= q_{0,1} \cdot q_{1,2} \cdot q_{2,3} \\ &= \frac{7}{32} \times \frac{1}{8} \times \frac{5}{32} \\ &= \frac{35}{8192} \end{aligned}$$

6.10.5 Modeling Criterion **S-8**

Criterion **S-8** can now be written as follows. Arrange the eight S-boxes $S_j, S_{j \bmod 8+1}, S_{(j \bmod 8+1) \bmod 8+1}$, $1 \leq j \leq 8$, so as to minimize the following quantity:

$$p = \max_{j=1,2,\ldots,8} q_{0,j} \cdot q_{1,j \bmod 8+1} \cdot q_{2,(j \bmod 8+1) \bmod 8+1} \tag{6.20}$$

Since the denominator of equation 6.20 is equal to at most 64^3 for 6×4 S-boxes taken three at a time, we can rewrite equations 6.17, 6.18 and 6.19 to avoid floating-point errors during the implementation of this criterion as follows. Compute:

$$\begin{aligned} Q_{0,j} &= \max_{c,d} 64 \times q_{0,j} \\ &= \max_{c,d} D_j(00cd11_2, 0) \\ &= \max\{D_j(3,0), D_j(7,0), D_j(11,0), D_j(15,0)\} && (6.21) \\ Q_{1,j} &= \max_{g,h} 64 \times q_{1,j} \\ &= \max_{c,d} D_j(11gh10_2, 0) \\ &= \max\{D_j(50,0), D_j(54,0), D_j(58,0), D_j(62,0)\} && (6.22) \\ Q_{2,j} &= \max_{k,m} 64 \times q_{2,j} \\ &= \max_{k,m} D_j(10km00_2, 0) \\ &= \max\{D_j(32,0), D_j(36,0), D_j(40,0), D_j(44,0)\} && (6.23) \end{aligned}$$

Example 6.13 *For adjacent S-boxes S_1, S_2 and S_3, equations 6.21, 6.22 and 6.23 yield the following values:*

$$\begin{aligned} Q_{0,1} &= \max\{14, 2, 2, 2\} = 14 \\ Q_{1,2} &= \max\{8, 8, 4, 4\} = 8 \\ Q_{2,3} &= \max\{0, 4, 6, 10\} = 10 \end{aligned}$$

The highest probability of success of an attack corresponds to the following quantity:

$$Q_{0,1} \times Q_{1,2} \times Q_{2,3} = 14 \times 8 \times 10 = 1120$$

Equation 6.20 is similarly transformed. Determine an arrangement of eight S-boxes $S_1, S_2, \ldots, S_8$ so as to minimize

$$P = \max_{j=1,2,\ldots,8} Q_{0,j} \cdot Q_{1,j \bmod 8+1} \cdot Q_{2,(j \bmod 8+1) \bmod 8+1} \tag{6.24}$$

The quantity P will be referred to as the *difference-score* of an arrangement of eight S-boxes S_1 to S_8.

Example 6.14 *We have found the maximum probability P using equation 6.24 for arranging S-boxes S_1, S_2, S_3. Denote the maximum probability for this triplet by P_1. Similarly, determine P_2 for the triplet S_2, S_3, S_4, p_3 for S_3, S_4, S_5, and so on, until P_8 for S_8, S_1, S_2. Compute $P^{(1)} = \max\{P_1, P_2, \ldots, P_8\}$. This is the difference-score for the arrangement $(S_1, S_2, \ldots, S_8)$.*

There are a total of 8! arrangements of all eight S-boxes. For each arrangement, compute difference-scores $P^{(i)}$ in the manner just mentioned, $2 \leq i < 8!$. Finally, determine the minimum of these difference-scores, equal to $\min_{i=1,2,\ldots,8!} P^{(i)}$. The arrangements of the eight S-boxes that correspond to this difference-score constitute the best possible arrangements that minimize the maximum probability of attack.

The results of implementing **S-8** are discussed in Chapter 7.

6.11 Summary

The various forms of symmetry of the constraints modeled in our CSP formulation of the S-box problem is summarized in Table 6.6.

Out of these forms of symmetries, the row, column and diagonal symmetry hold good only if the interchanges of rows (respectively, columns and quadrants) are made simultaneously. Otherwise some other constraints are violated and the resulting configuration is not an S-box. The rotational and bit inversion forms of

Constraint for Criterion	Row Symmetry	Column Symmetry	Diagonal Symmetry	Rotational Symmetry	Bit Inversion Symmetry
S-2					×
S-3					×
S-4	×				×
S-5		×			×
S-6			×		×
S-7				×	×

Table 6.6: Summary of Results on Symmetry of constraints modeling S-box Criteria

symmetry, however, yield an S-box that satisfy all constraints. The row, column and diagonal symmetry of the S-box may or may not impact its score. (This has to be either proved or disproved!) We have only verified this for DES S-boxes. However, the rotational and bit inversion symmetries do not affect the score of the S-box. We have proved these properties in this Chapter.

From the viewpoint of efficiency, an S-box results in an extra one due to row symmetry, one more due to column symmetry, a third due to diagonal, a fourth due to rotational and a fifth due to the bit inversion symmetry. In other words, one S-box results in $2^5 = 32$ S-boxes already available and therefore, a 32-fold speedup of S-box search. By adding symmetry breaking constraints to the solver, we should be able to prune further and avoid visiting these new S-boxes when they are eventually encountered in search.

Criterion **S-8** deals with multiple S-boxes taken three at a time. The objective of this criterion is to thwart differential cryptanalysis. An XOR table employed in Biham's work on differential cryptanalysis [11] is used to model this criterion.

This criterion is not implementable into the existing framework that yields one S-box at a time. In Chapter 9, we will discuss an alternative formulation that models **S-8** into a set of constraints. By doing this, the entire S-box formulation will be shown to be modeled strictly as a CSP.

Chapter 7
Performance Measures, Experiments and Results

We discuss four measures that will be used to evaluate the heuristics for the n-ary constraints developed in Chapters 4 and 5. One measure is concerned with the quality of S-boxes generated. The remaining three measures deal with execution efficiency and provide us with various kinds of information.

In this Chapter we discuss several experiments. To begin with, problems being studied for performance using various heuristics are summarized and labelled accordingly. The solver is run on small-sized CSP's to generate small-sized $n \times m$ S-boxes using the generate-and-test approaches to criteria **S-2** and **S-7**, verify completeness, and examine results. The summarized heuristics are studied through experiments and their performances, measured against the quality of S-boxes as well as the quantifications for efficiency developed in this chapter.

7.1 Summary of Problems

Algorithm 1 implemented as `Solver` using AC2001 is used to evaluate the following approaches to the modeling of S-box generation problems:

- The **S-2** criterion is translated to a single hard constraint based on the threshold τ, and the **S-7** criterion is implemented as a single n-ary constraint. This family of CSPs that employ the complete, non-incremental heuristics $V_S^{\phi,\tau}$, $VO_S^{\phi;\tau}$, $H_S^{\phi;\tau}$ and $HO_S^{\phi;\tau}$ for $n \times m$ S-boxes, is denoted by $DES_{S,\tau}^{n,m}$.

- The **S-2** criterion is implemented using the constraints employed by the incomplete, incremental heuristic $H_I^{\phi,\tau}$, in which **S-7** is implemented as a single n-ary constraint. This family of CSPs, for $n \times m$ S-boxes, is denoted by $DES_{I,\tau}^{n,m}$.

- The **S-2** criterion is implemented using the complete, incremental heuristic $H_C^{\phi,\tau}$, in which **S-7** is implemented as a single n-ary constraint. This family of CSPs, for $n \times m$ S-boxes, is denoted by $DES_{C,\tau}^{n,m}$.

- The **S-2** criterion is implemented using complete, incremental heuristics $V_{C_7}^{\Phi,\tau}$, $VO_{C_7}^{\Phi,\tau}$, $H_{C_7}^{\Phi,\tau}$, and $HO_{C_7}^{\Phi,\tau}$, where **S-2** and **S-7** are incrementally checked. This family of CSPs, for $n \times m$ S-boxes, is denoted by $DES_{C_7,\tau}^{n,m}$.

- The **S-2** criterion is implemented using complete, incremental heuristics $V_{AC_7}^{\Phi,\tau}$, $VO_{AC_7}^{\Phi,\tau}$, $H_{AC_7}^{\Phi,\tau}$, and $HO_{AC_7}^{\Phi,\tau}$, where **S-2** is incrementally checked and **S-7**, decomposed by projecting past assignments over domains of future variables. This family of CSPs, for $n \times m$ S-boxes, is denoted by $DES_{AC_7,\tau}^{n,m}$.

For brevity, throughout the remainder of this Chapter, Algorithm 1 that implements the solver of Section 2.7 will be referred to as $MAC2001(\tau)$, where τ is the threshold of the S-box score.

7.2 Performance Metrics

We now discuss what measures are compared while running experiments using the aforementioned heuristics, along with reasons for the measures considered. The following four metrics are used:

1. The quality metric of an S-box

2. A quantification of search points

3. CPU elapsed time, and

4. Number of *completely-filled* S-boxes generated

The quality metric of an S-box helps us compare how well a heuristic generates "better" quality S-boxes compared with another. Quantifying search points provides us information on how far into the search space each of the heuristics advances with time. The farther the advancement, the more efficient is the heuristic. The CPU elapsed time informs how long each heuristic took to generate a specified number of S-boxes. Finally, counting the number of completely-filled S-boxes is another efficiency metric.

The order in which the above metrics are listed should be noted. The score of an S-box is the most important metric because, we would like to obtain high-quality S-boxes as early as we can. The quantification of search points is next in order of preference. It gives us an idea of how "quickly" S-boxes can be obtained with the aforementioned heuristics. The CPU time gives us the same idea. However, the expression that quantifies the search points gives us more information about the nature of the search space that mere timing results do not. This fact will become apparent in the results of experiments. Finally, the number of S-boxes generated deals with completely-filled S-boxes alone, since counting partially assigned S-boxes do not make sense. This count only suggests speedup just as the CPU elapsed time does, and does not similarly provide information on the nature of search space the way the quantification results do.

We will actually encounter two kinds of information about the nature of search space. The first kind deals with "clusters" of S-boxes having the same "good" quality as adjudged by the quality metric. The second kind of information deals with spurts of "many" S-boxes being generated compared to a long duration between two S-boxes generated, as revealed by the quantification of search space and to a lesser extent, by CPU time and counting the number of solutions.

Let us now discuss the four performance metrics.

7.2.1 The Quality Metric for an S-box

The equation for the score $\sigma_X(\Phi)$ of an S-box Φ given by Equation 4.6 provides us with the quality metric for the S-box. For an $n \times m$ S-box, the maximum value of this score is equal to 2^{n-1} while the minimum value is equal to zero. An S-box Φ_1 is considered "better" than a second S-box Φ_2 if $\sigma_X(\Phi_1) < \sigma_X(\Phi_2)$.

7.2.2 A Measure of the Search Space

Instead of attempting an exhaustive coverage of the search space, the certitude of optimality is evaluated by measuring the fraction of search space that `Solver()` covers while generating S-boxes. Let us develop the concepts needed to formulate an expression for this metric. For simplicity, we assume that the domains for each variable in X are identical and equal to $\mathbb{Z}_d$.

Encoding for a Partially Assigned S-box Given a partial assignment involving variables in the set $X' = \{x_0, x_1, \ldots, x_{|X'|-1}\}$ with $X' \subseteq X$, an *encoding* E for an $n \times m$ partially assigned S-box is defined as:

$$S_p = \sum_{i=0}^{|X'|-1} \lambda^{-1}(x_{\pi(i)}) \cdot d^{|X'|-i-1} \tag{7.1}$$

where $d = 2^m$, and $\lambda : \mathbb{Z}_d \to \mathbb{Z}_d$ and $\pi : \mathbb{Z}_{|X'|} \to \mathbb{Z}_{|X'|}$ are permutation functions that determine value and variable ordering, respectively. π is discussed in section 5.5.2 while λ is discussed in section 7.3. λ^{-1} is the inverse permutation corresponding to λ on $\mathbb{Z}_d$. This encoding is easily extensible to a completely-filled S-box by setting $|X'| = |X|$, the number of variables in the S-box.

In simple words, each $n \times m$ S-box can be regarded as a radix-d number having $|X'|$ digits in that radix. The entry of this S-box for $\lambda^{-1}(x_{\pi(0)})$ is the most significant digit that varies least-frequently, while that for $x_{\pi(|X'|-1)}$ is the least significant digit that varies most-frequently.

We now state and prove two properties for S_p. For proving purposes, without loss of generality, let us define permutation functions $\lambda : Z_d \rightarrow Z_d$ and $\pi : Z_{|X'|} \rightarrow Z_{|X'|}$ as follows:

$$\begin{aligned} \lambda(k) &= k, \text{ where } 0 \leq k < d \\ \pi(k) &= k, \text{ where } 0 \leq k < |X'| \\ \text{so that } \lambda^{-1}(k) &= k \end{aligned}$$

Based on these definitions, Eq. 7.1 is rewritten as:

$$S_p = \sum_{i=0}^{|X'|-1} x_i \cdot d^{|X'|-i-1} \tag{7.2}$$

where $X' = \{x_0, x_1, x_2, \ldots, x_{|X'|-1}\}$.

Property 7.1 (Uniqueness) *S_p is unique to each assignment.*

Proof

The result follows at once if the number of variables in any two partial assignments differ. Consider two different (partial) assignments to the same number of variables in X':

$$\begin{aligned} A &= \langle (x_0, d_0), (x_1, d_1), \ldots, (x_{|X'|-1}, d_{|X'|-1}) \rangle, \\ A' &= \langle (x_0, d'_0), (x_1, d'_1), \ldots, (x_{|X'|-1}, d'_{|X'|-1}) \rangle \end{aligned}$$

where $d_i, d'_i \in Z_d$ and $d_i \neq d'_i$, whenever $0 \leq i < |X'|$. Let the encoding for A (A') be S_p (S'_p). We have to prove that $S'_p = S_p \Rightarrow A' = A$. From Eq. 7.2,

$$S_p = \sum_{i=0}^{|X'|-1} d_i \cdot d^{|X'|-i-1} \text{ and } S'_p = \sum_{i=0}^{|X'|-1} d'_i \cdot d^{|X'|-i-1}$$

For S_p and S'_p to be not unique, we should have $S'_p = S_p$, i.e.

$$\sum_{i=0}^{|X'|-1} d'_i \cdot d^{|X'|-i-1} = \sum_{i=0}^{|X'|-1} d_i \cdot d^{|X'|-i-1}$$

Equating the co-efficients, we obtain $d'_i = d_i$ whenever $0 \leq i < |X'|$. Therefore, $S'_p = S_p \Rightarrow A' = A$ establishing uniqueness.

Q.E.D.

From the encoding S_p, a (partial or complete) assignment is uniquely "retrieved" by repeatedly dividing S_p by d and collecting remainders that serve as the values assigned to the variables. The process terminates immediately when the dividend becomes zero. At this point the remaining variables are unassigned in the case of a partial assignment.

Property 7.2 (Strict Monotonicity) *S_p increases strictly monotonically as search progresses.*

Proof

Let the variables be represented by $X' = \{x_0, x_1, \ldots, x_j, x_{j+1}, \ldots, x_{|X'|-1}\}$. Let the domain of variable $x_{j+1} \in X'$ be $Z_d = \{d^{(0)}_{j+1}, d^{(1)}_{j+1}, \ldots, d^{(d-1)}_{j+1}\}$, with $d^{(k+1)}_{j+1} > d^{(k)}_{j+1}$ whenever $0 \leq k < d-1$. Then $d^{(k+1)}_{j+1} \geq d^{(k)}_{j+1} + 1$, and

$$d^{(k+1)}_{j+1} - d^{(k)}_{j+1} - 1 \geq 0 \tag{7.3}$$

Consider two *consecutive* (partial) assignments. We examine the case when the value to variable x_{j+1} changes from $d^{(k)}_{j+1}$ to the value $d^{(k+1)}_{j+1}$. We further consider the assignment to the remaining variables $x_{j+2}, \ldots, x_{|X'|-1}$ arising due to systematic search. In the pathological case, prior to x_{j+1} changing, these remaining variables had the maximum value in $\mathbb{Z}_d$, equal to $(d-1)$. When the value of x_{j+1} changes, the remaining variables now assume the minimum value in $\mathbb{Z}_d$, that is, 0. In other words, consider the following two assignments:

$$\begin{aligned} A_j &= \langle (x_0, d_0), (x_1, d_1), \ldots, (x_j, d_j), \\ &\quad (x_{j+1}, d^{(k)}_{j+1}), (x_{j+2}, d-1), \ldots, (x_{|X'|-1}, d-1) \rangle, \\ A_{j+1} &= \langle (x_0, d_0), (x_1, d_1), \ldots, (x_j, d_j), (x_{j+1}, d^{(k+1)}_{j+1}), (x_{j+2}, 0), \ldots, (x_{|X'|-1}, 0) \rangle, \end{aligned}$$

where $d_0, d_1, \ldots, d_j \in Z_d$.

Let $S^{(j)}_p$ ($S^{(j+1)}_p$) denote the encodings of A_j (A_{j+1}). Then using Eq. 7.2, we obtain:

$$
\begin{aligned}
S_p^{(j)} &= \sum_{i=0}^{|X'|-1} x_i \cdot d^{|X'|-i-1} \\
&= \sum_{i=0}^{j} d_i \cdot d^{|X'|-i-1} + d_{j+1}^{(k)} \cdot d^{|X'|-j-2} + \sum_{i=j+2}^{|X'|-1} (d-1) \cdot d^{|X'|-i-1} \\
S_p^{(j+1)} &= \sum_{i=0}^{|X'|-1} x_i \cdot d^{|X'|-i-1} \\
&= \sum_{i=0}^{j} d_i \cdot d^{|X'|-i-1} + d_{j+1}^{(k+1)} \cdot d^{|X'|-j-2} + \sum_{i=j+2}^{|X'|-1} 0 \cdot d^{|X'|-i-1}
\end{aligned}
$$

By subtraction,

$$
\begin{aligned}
S_p^{(j+1)} - S_p^{(j)} &= (d_{j+1}^{(k+1)} - d_{j+1}^{(k)}) \cdot d^{|X'|-j-2} - \sum_{i=j+2}^{|X'|-1} (d-1) \cdot d^{|X'|-i-1} \\
&= (d_{j+1}^{(k+1)} - d_{j+1}^{(k)}) \cdot d^{|X'|-j-2} - (d-1) \times \\
&\quad (1 + d + d^2 + \ldots + d^{|X'|-j-3}) \\
&= (d_{j+1}^{(k+1)} - d_{j+1}^{(k)}) \cdot d^{|X'|-j-2} - (d-1) \times \frac{d^{|X'|-j-2} - 1}{d-1} \text{ (if } d \neq 1) \\
&= (d_{j+1}^{(k+1)} - d_{j+1}^{(k)} - 1) \cdot d^{|X'|-j-2} + 1 \text{ (if } d \neq 1) \\
&\geq 1 \text{ from Eq. 7.3}
\end{aligned}
$$

The same result follows if $d = 1$, by setting this value for d directly into the first step. Since $S_p^{(j+1)} > S_p^{(j)}$, $S_p^{(j)}$ increases strictly monotonically.

Q.E.D.

Fraction of Search Space Let p, the fraction of the search space p $(0 \leq p < 1)$ covered by `Solver()`. p is approximated as follows. For an $n \times m$ S-box, the total number of enumerations is equal to $2^{m \times 2^n}$ and forms the denominator of p. The fraction p of search space for the 6×4 S-box is given by the following ratio:

$$p = \frac{S_p}{2^{m \times 2^n}} = \frac{\lambda^{-1}(x_{\pi(i)}) \cdot d^{|X'|-i-1}}{2^{m \times 2^n}}, \text{ from Eq. 7.1} \tag{7.4}$$

This is the metric that we will use in our experiments to compare performance of heuristics. However, we will express this metric in a more readable manner in the following paragraphs.

Heuristic	First Search Point S_{p_1} (in Hexadecimal), and Corresponding Fraction, a
$H_I^{\phi,16}$	$S_{p_1} =$ `033056659aa9cffc744728dbed1eb281300395a9566cfacfd7ed7b4e218214b8` $a =$ 0.012456321531011171706977161053827542615230725548026705570504434171742344721
$V_S^{\phi,16}$ $VO_S^{\phi,16}$	(No solutions found) (No solutions found)
$V_{C_7}^{\phi,16}$	$S_{p_1} =$ `03569acf7421edb83065a9fc4712de8b3065acf94d8b712e09f3c05a824d0000` $a =$ 0.013040233276235570484942204477017154420405709642663933645624638940629054168
$V_{AC7}^{\phi,16}$	$S_{p_1} =$ `3569acf7421edb83065a9fc4712de8b3065acf94d8b721e0ca960351b478000` $a =$ 0.013040233276235570484942204477017154420405709642663933683860530915394054745
$VO_{C_7}^{\phi,16}$	$S_{p_1} =$ `3569acf7421edb83065a9fc4712de8b3065acf94d8b712e053a000000000000` $a =$ 0.013040233276235570484942204477017154420405709642663933645624638940629054168
$VO_{AC7}^{\phi,16}$	$S_{p_1} =$ `3569acf7421edb83065a9fc4712de8b3065acf94db8712e05a9c3061b4e0000` $a =$ 0.013040233276235570484942204477017154420405709642664403467543665769604504057
$H_S^{\phi,16}$, $HO_S^{\phi,16}$	$S_{p_1} =$ `033056659AA9CFFC74472112EDDEB88B30036556CFFC9AA9DEED8BB821127447` $a =$ 0.012456321531011171706977135898667979325824166971089322438578599427830606050
$H_{C_7}^{\phi,16}$	$S_{p_1} =$ `033056659AA9CFFC74472112EDDEB88B300369AF5CC5F69A4DB8100000000000` $a =$ 0.012456321531011171706977135898667979325824167732326887336844001622173067893
$H_{AC7}^{\phi,16}$	$S_{p_1} =$ `33056659aa9cffc74472112eddeb88b300369acf5cf5a964d7e82d700000000` $a =$ 0.012456321531011171706977135898667979325824167730683235082199556313454298076
$HO_{C_7}^{\phi,16}$	$S_{p_1} =$ `033056659aa9cffc74472112eddeb88b300369a6f55f9ccaed70000000000000` $a =$ 0.012456321531011171706977135898667979325824167726576701853008982422599743978
$HO_{AC7}^{\phi,16}$	$S_{p_1} =$ `33056659aa9cffc74472112eddeb88b300369af5cc5fa968bedd0000000000` $a =$ 0.012456321531011171706977135898667979325824167732326887499378089911418120883

Table 7.1: First search point generated by solvers employing various heuristics, on ordered domains of variables

First Search Point, and Offset Let S_{p_1} represent the first-generated S-box by the solver. Let a represent the fraction of search space covered by the solver when the first solution is obtained. The value for a is obtained by substituting the value of S_{p_1} into equation 7.4:

$$a = \frac{S_{p_1}}{2^{m \times 2^n}} \tag{7.5}$$

For all solutions different from the first, define a *search offset* r as follows:

$$r = p - a = \frac{S_p - S_p^1}{2^{m \times 2^n}} \tag{7.6}$$

Note that $|r| < 1$. (For the first search point, $r = 0$.) The remaining subsections analyze the variation in $p = a + r$ with time for the heuristics used in `Solver()`.

Table 7.1 lists out the value of S_{p_1} for each heuristic. The incomplete heuristic $H_I^{\phi,16}$ shows a value different from all of the other, complete heuristics, which is to be expected. Even for complete heuristics, the straight-line variable ordering shows an S_{p_1} value different from the zig-zag ordering, once again an expected result.

7.2.3 CPU Elapsed Time

Timing results have been recorded for every S-box generated (including partially assigned S-boxes). These timings are the CPU execution times and are output by the function `getrusage(RUSAGE_SELF, &time)` where `time` is the elapsed time (in seconds) since the experiment started. This function is called twice, once at start to record the elapsed CPU time at the first search point, and thereafter, after each search point is encountered, often at each minute. A comparison of CPU times by the different heuristics provides information related to speed-up. This is done for reporting purposes only. No further analysis is carried out on this metric. In the experimental results reported in this Chapter, the colum “Time (seconds)” or “Time (hrs)” always refers to the CPU elapsed time.

7.2.4 Number of Completely-filled S-boxes

Whenever an S-box with all entries is obtained, a count of the number of solutions is incremented by 1. After each minute, along with the CPU time, the number of solutions is also reported. A comparison of the number of complete S-boxes generated so far, against each heuristic, provides information on efficiency of the heuristic. Once again, this is done for reporting purposes only. No further analysis is made on this metric.

7.3 Random Permutation of Domains

To study the effect of domain-ordering on heuristic performance, the search space is shuffled by randomly permuting the domains of each variable. Procedure `Permute` from [25] permutes an ordering of integers π using random seed s, and, for reproducibility purposes, is provided below. The `drand48()` function in `Permute` is the one provided in standard GNU C library. Also, to allow the replication of the reported experiments, the seed is set at start with the help of the function `seed48([1000,0,0])`. The `swap()` function interchanges two integers.

7.4 Setup

The hardware environment consists of an Intel Pentium Core-2 Duo 3-GHz CPU and 3.3 GB RAM. GNU/Linux Ubuntu 9.04 is the operating system. Binary constraints are precompiled for S-box criteria **S-3**, **S-4**, **S-5**, and **S-6** (Section 2.3.1).

```
Procedure Permute($\pi$, $s$)
   inputs : An ordering $\pi$ of integers, and a seed $s$.
   output: A permutation of $\pi$
1  begin
2      Let $\pi = (d_0, d_1, \ldots, d_{|\pi|-1})$ ;
3      seed48($s$) ;
4      for $i \leftarrow 0$ to $(|\pi| - 1)$ do
5          $p \leftarrow \lfloor (|\pi| - i) \times$ drand48() $\rfloor$ ;          /* $0 \leq p < |\pi| - i$ */
6          swap($d_i, d_{p+i}$);
```

These constraints are then input to the solvers implementing the aforementioned heuristics for criteria **S-2** and **S-7**.

The experiments are broadly classified into the following types:

1. Efficiencies for small-sized S-boxes such as 4×4 and 5×3 S-boxes, discussed in Section 7.5.

2. Generation of complete 6×4 S-boxes, discussed in Section 7.6. The experiments have been run for a duration ranging from 5 hours to 4 days to capture various information regarding complete S-box generation.

3. Comparison of performance of heuristics for 6×4 S-box generation. This involves finding how far in the search space each heuristic has advanced up to. The measure of Section 7.2.2 is used in the comparison. Here, partial S-boxes are sampled each minute and included in the performance plots. The experiments have been run for two days, and the results are discussed in Section 7.7. The quality metric (the score) is thresholded by a maximum value τ in each experiment (refer section 4.3), and observations on the quality of generated S-boxes is made.

7.5 Efficiencies for Small-Sized S-boxes

In this section we evaluate the scores $\sigma_X(\Phi)$ of each of the eight published DES S-boxes Φ. Next, we attempt to generate all 4×2 S-boxes and also, examine those of Simple DES [62]. Finally, an attempt is made to generate all 5×3 S-boxes to examine the duration of search.

S-**box,** Φ	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
Score, $\sigma_X(\Phi)$	14	14	14	10	14	12	18	12

Table 7.2: Scores obtained for existing 6×4 S-boxes of DES

Constraint Combinations	**Time (seconds)**	**# of** S**-boxes**	**Score-wise breakup**			
			Score 8	**Score 6**	**Score 4**	**Score 2**
No constraints	136228.906250	4294967296	3931260	517882560	3496729600	276422720
S-3 only	35.029600	331776	11904	153600	166272	0
S-4 only	0.000089	4	4	0	0	0
S-5 only	6.410940	65536	7936	45056	12544	0
S-6 only	13214.516602	429981696	2103616	91728896	323934912	12214272
S-3, **S-5**	0.433693	4096	384	2048	1664	0
S-3, **S-6**	5.224500	46656	6240	22272	18144	0
S-5, **S-6**	2.085620	20736	4160	13312	3264	0
S-3, **S-5**, **S-6**	0.165739	1600	224	768	608	0

Table 7.3: Statistics of 4×2 S-boxes generated by $MAC2001(\tau)$ to satisfy combinations of DES criteria

7.5.1 Evaluation of DES S-boxes S_1 to S_8

We employ Matsui's metric to score the eight DES S-boxes of Figure 1.1, proposed by IBM. The scores we have found are reported in Table 7.2.

Observation S-box S_7 possesses the (not-so-good) score of 18 and S-box S_4, the (best) score of 10. In addition, a number of S-boxes possess a score of 14. No S-box with a score of 8 was found during the manual construction. Interestingly, the last S-box S_8 used in breaking DES [11], yielded a "second-best" score of 12.

In general, the maximum value τ of the score of an $n \times m$ S-box is equal to 2^{n-1}, and is the value used for small values of n as the experiments suggest. For large-sized S-boxes such as 6×4, the maximum value of the score is considered equal to 16 ($= \frac{2^6}{4}$) and not $2^{6-1} = 32$ since, there are too many 6×4 S-boxes generated having score equal to 16 and we would like to study generation of (better) S-boxes with smaller scores.

7.5.2 Problem $DES^{4,2}_{S,8}$: Generation of 4×2 S-boxes

This problem generates 4×2 S-boxes (16 variables, at domain size 4). Criterion **S-7** no longer applies. Not all criteria **S-3**, **S-4**, **S-5** and **S-6** result in solutions when applied together. Table 7.3 reports results obtained using some combinations of criteria that yield solutions. The threshold τ assumes the maximum value of $2^{4-1} = 8$.

Total time (seconds)	**Total number of S-boxes**	**Score-wise breakup**		
		Score 16	**Score 12**	**Score 8**
14.2659	32,640	25728	3456	3456

Table 7.4: The scores of 5×3 S-Boxes generated by the model, with criteria **S-5** and **S-6** relaxed

The S-boxes of Simple DES [62]

Simple DES is proposed in [62] and employs two 4×2 S-boxes S_0 and S_1. Our model reports that both S-boxes S_0 and S_1 do not satisfy DES criteria **S-3** to **S-6**. Applying criterion **S-2** yielded a score equal to 5 for S-box S_0. In general, all entries in the linear approximation table for an S-box, and therefore the score, should be even. The reason for the odd-numbered score in the case of S-box S_0 is that this S-box has two identical entries in row 3 (resulting in criterion **S-3** not being satisfied). S-box S_1 yielded a score of 6. Two entries in row 2 of S-box S_1 are identical, resulting in the presence of an odd-numbered entry equal to 5 in its linear approximation table (and criterion **S-3** not being satisfied).

Conclusion: This approach on problem $DES^{4,2}_{S,8}$ generates 4×2 S-boxes only when some of the DES criteria **S-2** to **S-7** are relaxed. Results on combinations of satisfied criteria, generation times and number of S-boxes are reported in Table 7.3.

7.5.3 5×3 S-boxes

This problem, specified as $DES^{5,3}_{S,16}$, generates 5×3 S-boxes (32 variables, each with domain cardinality equal to 8). Criteria **S-5** and **S-6** are relaxed. The threshold assumes a maximum value of $2^{5-1} = 16$. The solver could find all solutions and terminate. Table 7.4 reports S-box generation times and number of S-boxes for different scores, with a total of 32,640 S-boxes generated.

Conclusion: The approach of running `Solver()` on problem $DES^{5,3}_{S,16}$ generates 5×3 S-boxes to satisfy all remaining constraints when criteria **S-5** and **S-6** are relaxed. No 5×3 S-boxes of scores 6, 4 and 2 were found.

7.6 Experimental Results for 6×4 S-box Generation

In the earlier problems for generating smaller-sized S-boxes, the complete heuristic was used. For 6×4 S-boxes, however, we have formulated several alternative heuristics for criteria **S-2** and **S-7** to improve search speed and/or S-box quality. As such, a whole section is devoted to 6×4 generation.

Time (hrs)	**Search space increment, r**	**# of S-boxes** ($\sigma_X(\Phi) = 16$)
1	1.1980×10^{-49}	4
2	2.1725×10^{-48}	14
3	4.2091×10^{-48}	15
4	4.2091×10^{-48}	26
5	6.1340×10^{-48}	40

Table 7.5: $MAC2001(16)$ on Problem $DES^{6,4}_{S,16}$ – Performance statistics

7.6.1 Problem $DES^{6,4}_{S,16}$: Complete, Non-incremental Heuristics

We report the results of executing `Solver()` employing the following complete, non-incremental heuristics: $H^{64,\tau}_S$, $HO^{64,\tau}_S$, $V^{64,\tau}_S$ and $VO^{64,\tau}_S$. The former two heuristics are the optimized and non-optimized versions employing default variable-ordering, while the latter two heuristics are the corresponding versions that employ even/odd variable-ordering.

S-boxes generated by `Solver()`, for each heuristic, are sampled each minute and the fraction of search space offset is determined using Equation 7.6. Table 7.1 records the values for the fraction of search-space a traversed prior to the first S-box. The values differ with the variable-ordering heuristic employed, otherwise, they are identical for all heuristics within that ordering (as expected).

Table 7.5 reports the search time (seconds), the increment r of the fraction of search space covered from the partial S-box generated during the first minute, and the number of S-boxes with all entries filled, obtained in these time frames, for heuristics $H^{\phi,16}_S$ and $V^{\phi,16}_S$ that employ no optimization.

All searches with this approach have so far resulted in 6×4 S-boxes with score $\sigma_X(\Phi)$ equal to 16. As Table 7.5 reports, the first four S-boxes having score 16 were generated after about an hour of search commencement. Large wait-times were visible between S-box generations such as, for example, between 14 and 15 solutions.

Specifying the maximum score τ equal to 16 should enable $MAC2001(16)$ to generate S-boxes with "better" scores (values of $\sigma_X(\Phi)$ less than τ). However we report that S-boxes with "better" scores of 14, 12, 10, 8, and so on did not surface from this approach in the stipulated time-frames. We also report that experiments with $MAC2001(\tau)$ for $\tau = 14, 12, 10$ and 8 did not yield 6×4 S-boxes for these thresholds in the five-hour time-frame used for running $MAC2001(16)$.

The issue addressed in the next two problems is the improvement of search speed of $MAC2001(\tau)$ over $DES^{6,4}_{S,\tau}$.

Time (hrs)	**Search space increment,** r	**# of** S**-boxes** ($\sigma_X(\Phi) = 16$)
1	1.0216×10^{-44}	20786
2	2.6504×10^{-44}	35957
3	9.1542×10^{-44}	49110
4	9.9395×10^{-44}	80933
5	1.0615×10^{-43}	94069

Table 7.6: Algorithm $MAC2001(16)$ on Problem $DES^{6,4}_{C,16}$ – Performance statistics

Time (hrs)	**Search space increment,** r	**# of** S**-boxes** ($\sigma_X(\Phi) = 16$)
1	1.6929×10^{-44}	24524
2	9.6042×10^{-44}	43462
3	9.9080×10^{-44}	68668
4	9.9956×10^{-44}	93523
5	1.1456×10^{-43}	108043

Table 7.7: $MAC2001(16)$ on Problem $DES^{6,4}_{C_7,16}$ – Performance statistics

$MAC2001(16)$ on Problems $DES^{6,4}_{C,\tau}$, for various thresholds τ (Soft Constraint Decomposition)

The results of running $MAC2001(\tau)$ on the family of problems $DES^{6,4}_{C,\tau}$ are reported in Table 7.6, with $\tau = 16$.

Observations A comparison with the results of Table 7.5 suggests that using the formulation of problem $DES^{6,4}_{C,\tau}$ indeed speeded up the search for S-boxes when $\tau = 16$, where the marginal coverage of the search space grows with a factor of 10^5. This model is complete, finding all solutions the way the model involving the formulation of problem $DES^{6,4}_{S,\tau}$ does.

This approach shares the drawback as with problems $DES^{6,4}_{S,\tau}$, of its inability to generate S-boxes with scores of 14, 12, 10 and 8 in reasonable time-frames, when $\tau = 16$, or when $\tau = 14, 12, 10, 8$.

Problem $DES^{6,4}_{C_7,\tau}$, for various thresholds τ

The results of running Algorithm $MAC2001(\tau)$ on this problem are reported below for $\tau = 16$.

A comparison with the results of Table 7.5 suggests that using the formulation of problem $DES^{6,4}_{C,\tau}$ indeed speeded up the search for S-boxes when $\tau = 16$, where the marginal coverage of the search space grows with a factor of 10^5. This model is complete, finding all solutions the way the model involving the formulation of problem $DES^{6,4}_{S,\tau}$ does.

A comparison of this table with the results of Table 7.5 suggests that using the formulation of $DES^{6,4}_{C_7,16}$ indeed speeded up the search for S-boxes when $\tau = 16$, where the marginal coverage of the search space grows with a factor of 10^5, while comparing with Table 7.6 reveals a marginal factor of 1.08.

Algorithm $MAC2001(\tau)$ on problem $DES^{6,4}_{C_7,\tau}$ possesses the same disadvantage as on problems $DES^{6,4}_{S,\tau}$ and $DES^{6,4}_{C,\tau}$, namely, that even when $\tau = 16, 14, 12, 10, 8$, the model does not generate S-boxes with lesser scores.

The speedup is observed in the plot of Figure 7.1 generated from a four-day run of the above experiments using formulations of problems $DES^{6,4}_{C,16}$ and $DES^{6,4}_{C_7,16}$. The plot displays "jumps" at the points at which a number of variables get reassigned. One such jump, not evident in the plot, occurred nearly one-and-a-half hours after start of the experiment, from an increment- point of 2.6532×10^{-44} (after 38062 S-boxes were generated) to a value of 8.3333×10^{-44} (when the next S-box was obtained). Frequent, gradual jumps were visible between 24 and 32 hours. The more the number of jumps, the more the search space is uncovered. Also shown in the plot is the first approach using the formulation to problem $DES^{6,4}_{S,16}$. The first approach did not exhibit any such jumps in the two days that it was run. These jumps would have presumably occurred after a long duration.

Conclusion: The approach using the formulation of problem $DES^{6,4}_{C_7,\tau}$ to generate 6×4 S-boxes results in the fastest model we currently have, compared to those resulting from problems $DES^{6,4}_{S,\tau}$ and $DES^{6,4}_{C,\tau}$.

None of these models have yielded 6×4 S-boxes having scores below 16 so far. The issue addressed in the next problem is generation of S-boxes with scores below (and including) τ.

Problem $DES^{6,4}_{I,\tau}$, for various thresholds τ

The approach using the formulation for problem $DES^{6,4}_{I,\tau}$ is used to generate S-boxes with different thresholds $\tau = 16, 14, 12, 10, 8$ The experiment is run separately on each of these τ-values for a five-hour duration. The following observations are made in this time-frame.

- *This approach yields S-boxes with scores τ and $(\tau - 2)$, and no more, when* $\tau = 16, 14, 12, 10$. The results in Tables 7.8 to 7.11 for each τ report the measure of search-space covered, and the number of S-boxes generated in each hour over the stipulated duration.

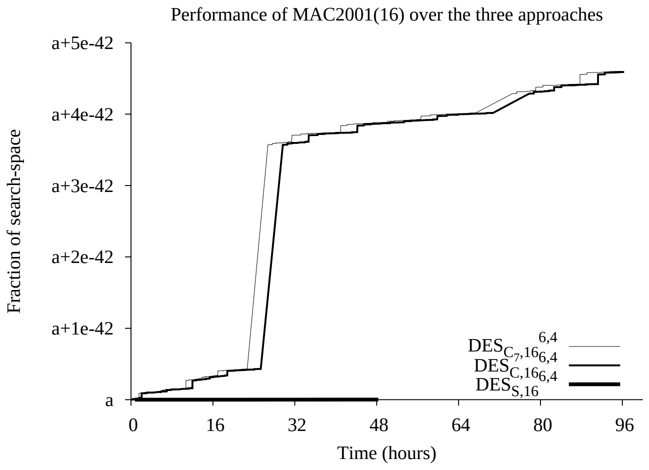


Figure 7.1: Fraction of search space as a function of search time (hours), covered by Algorithm $MAC2001(16)$ on Problems $DES^{6,4}_{S,16}$, $DES^{6,4}_{C,16}$, and $DES^{6,4}_{C_7,16}$.

Time (hrs)	**Search space offset, r**	**# of S-boxes**	
		$\sigma_X(\Phi) = 16$	$\sigma_X(\Phi) = 14$
1	8.9771×10^{-46}	44698	11952
2	9.3263×10^{-46}	95205	31890
3	9.0561×10^{-44}	145929	46247
4	9.0585×10^{-44}	194165	62274
5	9.0906×10^{-44}	240745	79167

Table 7.8: Approach using problem $DES^{6,4}_{I,16}$ formulation – Performance statistics

- *S-boxes with different scores appear immediately at a number of points in the search space.* Note the different points in search-space signified by the different values for r in Table 7.8, where $\sigma_X(\Phi) = 16$, compared with those in Tables 7.5, 7.6 and 7.7. Note also, how closely the solutions for score $(\tau - 2)$ follow those for score τ, given τ. This finds S-boxes having scores $(\tau - 2)$ and τ in parts of the search tree different from those explored by Algorithm $MAC2001(16)$ on Problems $DES^{6,4}_{S,16}$, $DES^{6,4}_{C,16}$, and $DES^{6,4}_{C_7,16}$.

- *The S-boxes generated by the approach using the formulation for Problem $DES^{6,4}_{I,\tau}$, having score $\sigma_X(\Phi) = (\tau - 2)$, need not be identical to those for problem $DES^{6,4}_{I,\tau-2}$, where $\tau = 16, 14, 12$.* Among Tables 7.8, 7.9, 7.10, and 7.11, this can be seen in any two under the column for r, for the same

Time	**Search space**	**# of S-boxes**	
(hrs)	**offset, r**	$\sigma_X(\Phi) = 14$	$\sigma_X(\Phi) = 12$
1	5.6792×10^{-47}	21584	3602
2	8.9688×10^{-44}	50235	6827
3	9.0033×10^{-44}	83017	12496
4	9.0559×10^{-44}	121126	18748
5	1.0650×10^{-43}	157850	23692

Table 7.9: Approach using problem $DES^{6,4}_{I,14}$ formulation – Performance statistics

	$\sigma_X(\Phi) = 12$		$\sigma_X(\Phi) = 10$	
Time (hrs)	**Search space offset, r**	**# of S-boxes**	**Search space offset, r**	**# of S-boxes**
1	1.7846×10^{-44}	11056	1.7965×10^{-44}	103
2	6.1985×10^{-44}	23160	6.2677×10^{-44}	163
3	3.0668×10^{-43}	37652	3.0668×10^{-43}	418
4	3.1389×10^{-43}	50742	3.1389×10^{-43}	850
5	3.1424×10^{-43}	62293	3.1424×10^{-43}	1041

Table 7.10: Approach using Problem $DES^{6,4}_{I,12}$ formulation – Performance statistics

	$\sigma_X(\Phi) = 10$		$\sigma_X(\Phi) = 8$	
Time (hrs)	**Search space offset, r**	**# of S-boxes**	**Search space offset, r**	**# of S-boxes**
1	3.5594×10^{-44}	8562	3.5594×10^{-44}	3583
2	5.7281×10^{-41}	17827	6.2206×10^{-41}	4999
3	6.4607×10^{-41}	27875	6.4607×10^{-41}	7836
4	6.8814×10^{-41}	37875	6.8814×10^{-41}	10883
5	1.0300×10^{-40}	47671	1.0300×10^{-40}	13602

Table 7.11: Approach using Problem $DES^{6,4}_{I,10}$ formulation – Performance statistics

score $\sigma_X(\Phi)$. For example, the values of r in Table 7.8 and Table 7.9 when $\sigma_X(\Phi) = 14$ differ under the same time-row.

- *The experiment generates "better" S-boxes compared to the earlier experiment involving formulations for problems $DES^{6,4}_{S,16}$, $DES^{6,4}_{C,16}$ and $DES^{6,4}_{C_7,16}$.* Even

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	3	5	6	9	10	15	12	7	4	14	13	2	1	8	11
1	3	0	6	5	10	9	12	15	4	7	13	14	1	2	11	8
2	3	15	0	12	5	6	9	10	4	8	7	11	14	13	2	1
3	0	12	3	15	9	10	5	6	7	11	4	8	2	1	14	13

Figure 7.2: A 6×4 S-box with score 8, generated by Algorithm MAC2001(10) applied over Problem $DES^{6,4}_{I,10}$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	3	5	6	9	10	15	12	7	4	14	13	2	1	8	11
1	3	0	6	5	10	9	12	15	4	7	13	14	1	2	11	8
2	3	15	0	12	5	6	9	10	4	8	11	2	14	13	7	1
3	0	12	3	15	9	10	5	6	7	11	4	8	2	1	14	13

Figure 7.3: A 6×4 S-box with score 10, generated by Algorithm MAC2001(10) applied over Problem $DES^{6,4}_{I,10}$

when $\tau = 16$, this experiment yielded S-boxes in times below the approaches for the earlier problems, particularly for Problem $DES^{6,4}_{S,16}$. For example, in one hour, only four S-boxes were obtained using Algorithm $MAC2001(16)$ on Problem $DES^{6,4}_{S,16}$, all having score 16. In comparison, the same algorithm on Problem $DES^{6,4}_{I,16}$ yielded 56649 S-boxes, with score equal to 16 and 14.

Algorithm $MAC2001(10)$ on Problem $DES^{6,4}_{I,10}$ yielded S-boxes having scores of 10 and 8. A sample S-box with a score equal to 10 is shown in Figure 7.3, and one with score 8, in Figure 7.2.

S-boxes below score 8 have not been obtained in the stipulated time-frame. Based on Matsui's S-box quality metric, S-boxes with a score of 8 are superior to those specified for DES, with the "best" DES S-box having a score equal to 10 as Table 7.2 reports.

Summary

Applying Algorithm $MAC2001(\tau)$ over Problem $DES^{6,4}_{I,\tau}$ additionally yielded "better" S-boxes having score equal to at most $(\tau - 2)$, when $\tau = 16, 14, 12, 10$. However, those S-boxes are not identical to the ones generated by applying Algorithm $MAC2001(\tau - 2)$ over Problem $DES^{6,4}_{I,\tau-2}$. The search-point is seen to differ, suggesting the incomplete nature of this model. However the time taken to generate S-boxes is less compared to that of the complete model obtained by applying Algorithm $MAC2001(\tau)$ over Problem $DES^{6,4}_{S,\tau}$. At the same time, the latter model, and its improvements (Algorithm $MAC2001(\tau)$ on Problems $DES^{6,4}_{C,\tau}$ and $DES^{6,4}_{C_7,\tau}$) have not been as capable of generating "better" S-boxes the way the former performed in the stipulated time-frame.

7.7 Comparisons Between Heuristics

In this section, a comparison of the following heuristics is made from the viewpoint of performance:

1. Non-incremental heuristics

2. Soft Constraint Decomposition Heuristic for **S-2** and Incremental Check for **S-7**, namely, heuristic ${H_{C_7}}^{\phi}$

3. Decomposition of the COUNT constraint for **S-7** by projection onto domains of future variables, which is heuristic ${H_{AC7}}^{\phi}$.

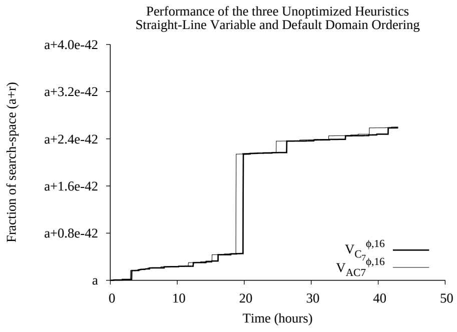
In this section, the heuristics will be compared for performance of 6×4 S-box generation against the following parameters.

1. Subsection 7.7.1 demonstrates the effect of introducing the condition for no arc-consistency check upon domain wipe-out, which is the optimization discussed in subsection 5.5.1.

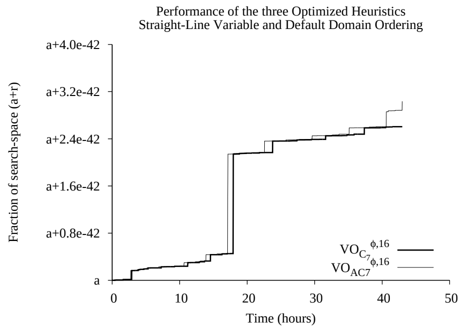
2. Subsection 7.7.2 compares heuristic performance for Straight-line variable ordering versus zig-zag variable ordering, discussed in subsection 5.5.2

Each subsection, compares complete heuristics over the default ordering of domains of variables, and domains permuted using the random permutation discussed in Section 7.3 by the procedure `Permute`. For 6×4 S-boxes, the default ordering of the domains is the increasing order, namely, the set $\{0, 1, 2, \ldots, 15\}$. The setup for the random permutation of domains has been discussed in Section 7.3.

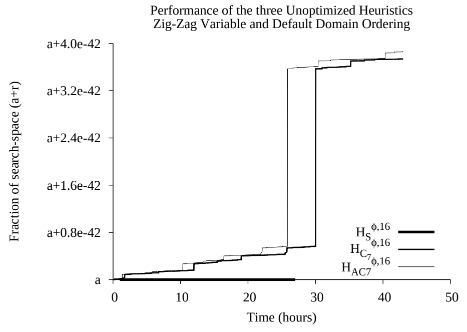
In the experiments, the threshold of the score sought is $\tau = 16$, a value better than the maximum score of 18 of DES S-box S_7.



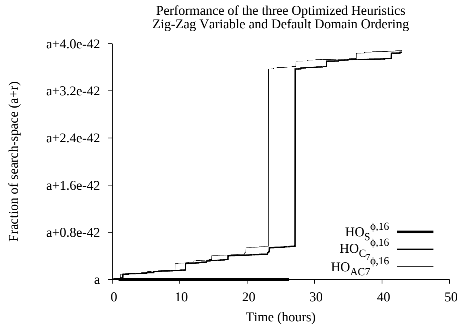
(a)



(b)



(c)



(d)

Figure 7.4: Performance of unoptimized and optimized heuristics. Heuristics V_S^{ϕ} and VO_S^{ϕ} have not generated any S-boxes in the stipulated time-frame and no curve is visible for these heuristics in Plots (a) and (b)

7.7.1 Performance of Unoptimized versus Optimized Heuristics

Eight plots are provided in this analysis in which unoptimized heuristics and their optimized variants are separately compared. Four of these plots are for normal ordering of domains of variables while the remaining four are for permutation of domains.

Default Domain-Ordering

Figure 7.4 displays the plots of heuristics H_S^{ϕ}, HO_S^{ϕ}, V_S^{ϕ}, and VO_S^{ϕ} for default ordering of values to variables.

The plots are similar in shape indicating that all of these heuristics traverse the same path in the search tree. The plots, as expected, are monotically increasing.

The almost-horizontal lines indicate that almost all values assigned to variables do not change except for the last few of these. When a vertical-line (jump) is encountered, it means that values assigned to the last several variables have changed.

Straight-Line Variable Ordering Notice that the non-incremental heuristics V_S^ϕ and its optimized variant VO_S^ϕ has not produced any S-boxes during the two-day run of the experiment, and is not present in the first two plots.

In the case of incremental checking, optimization results in a *slight* improvement of $VO_{C_7}^\phi$ (VO_{AC7}^ϕ) over the $V_{C_7}^\phi$ (V_{AC7}^ϕ). In particular, near the 20-hour mark, $VO_{C_7}^\phi$ depicts a 5.3-percent increase over $V_{C_7}^\phi$. VO_{AC7}^ϕ exhibits a 4.6-percent increase over V_{AC7}^ϕ. For both unoptimized and optimized heuristics, the increase becomes more prominent as time progresses and as the heuristics advance further into the search space.

Zig-Zag Variable Ordering The non-incremental heuristic H_S^ϕ and its optimized variant HO_S^ϕ has generated 384 S-boxes during the two-day run of the experiment.

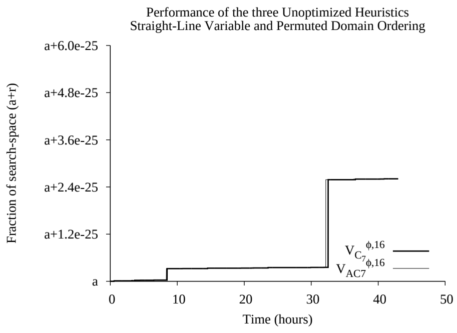
Without optimization, $H_{AC7}^{\phi,16}$ consistently shows a 14-percent performance increase over $H_{C_7}^{\phi,16}$. With optimization in place, the performance of respective heuristics $HO_{AC7}^{\phi,16}$ over $HO_{C_7}^{\phi,16}$ has slightly improved to 15%.

One can conclude that projecting the COUNT constraint over future variables, reducing their domains during the process, exhibits superior performance over incremental checking of the COUNT constraint regardless of whether optimization is present or not in both cases. The optimization of Table 5.9 results in further speedup due to reduction in calls to the function `EstablishFullAC`.

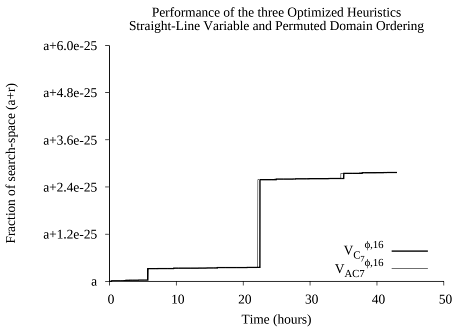
Permuted Domain-Ordering

Figure 7.5 depicts the performance characteristics of unoptimized heuristics $V_{C_7}^\phi$, V_{AC7}^ϕ, and optimized heuristics $VO_{C_7}^\phi$ and VO_{AC7}^ϕ, when the domains of variables are randomly permuted using the seed of 1000.

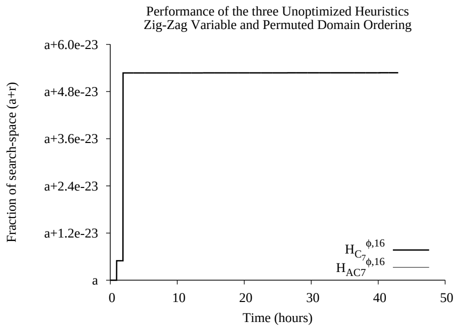
Notice that for this particular permutation of domains, the incremental heuristic that projects past assignments on to domains of future variables exhibits no significant improvement over the incremental checking heuristic, regardless of whether variables are ordered by straight-line or zig-zag approach.



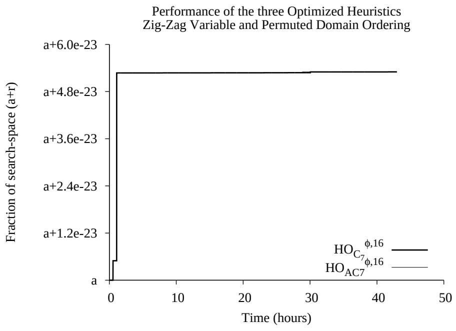
(a)



(b)



(c)



(d)

Figure 7.5: Performance of unoptimized and optimized heuristics for permuted domains

Straight-Line Variable Ordering The optimized versions of both of the aforementioned heuristics are seen to exhibit a 46-percent increase over the unoptimized versions for this particular permutation. Therefore for this variable ordering, the optimization by preventing further consistency-checking in case a domain wipe-out occurred, appears promising when domains are randomly permuted.

Zig-Zag Variable Ordering The performance of the optimized heuristic employing this form of variable ordering differs from that of the unoptimized heuristic by a very small amount, about 1.5 percent. However, the heuristics due to this ordering exhibit a jump within the second minute, over the solution space. Moreover, the ensuing horizontal line suggests that a lot of solutions differing only in the values of the last few variables is found using this permutation.

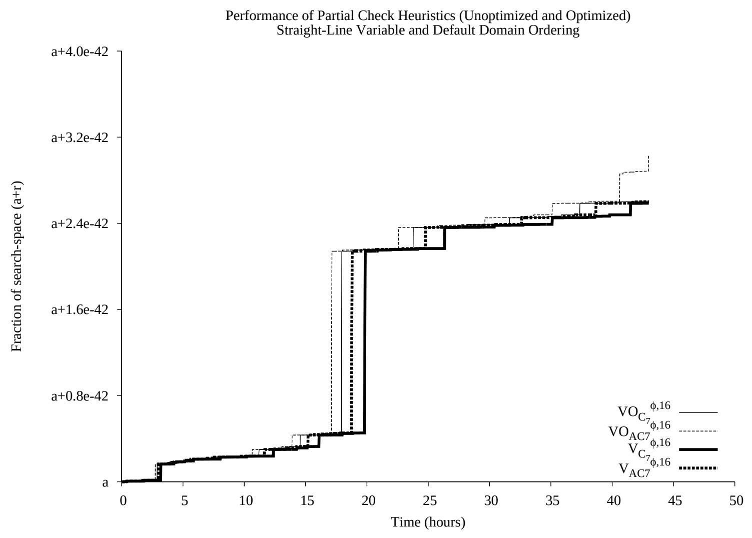


Figure 7.6: Performance of heuristics employing Straight Line Variable Ordering for ordered domains

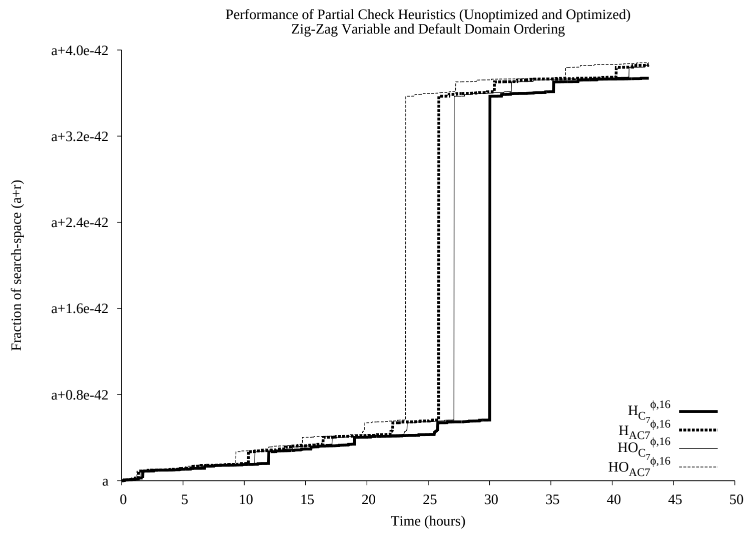


Figure 7.7: Performance of heuristics employing Zig-Zag Variable Ordering for ordered domains

7.7.2 Performance of Heuristics using Straight-Line and Zig-Zag Variable Ordering

Figure 7.6 depicts the performance of all incremental heuristics (unoptimized and optimized), using straight line variable ordering, and Figure 7.7, using zig-zag

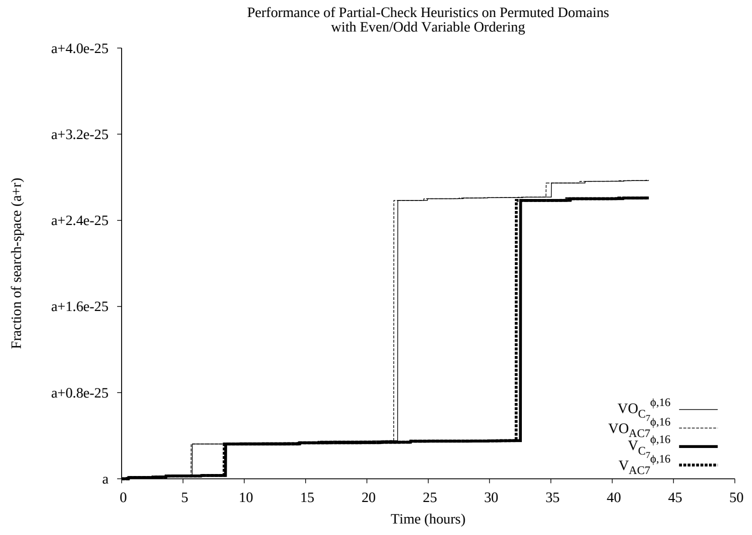


Figure 7.8: Performance of heuristics employing Straight Line Variable Ordering for permuted domains

variable ordering, both for ordered domains. Figure 7.8 and 7.9 exhibit the plots for permuted domains.

Default Domain-Ordering

In the plot of Figure 7.6 employing straight line variable ordering, each heuristic appears to exhibit a 10-percent increase in efficiency against the other as per the following ordering: $VO^{\phi}_{AC7} \geq VO_{C_7}\phi \geq V^{\phi}_{AC7} \geq V^{\phi}_{C_7}$. The plot suggests that the optimized version of $VO^{\phi}_{C_7}$ is more efficient than the unoptimized version of V^{ϕ}_{AC7}.

The plot of Figure 7.7 employing zig-zag ordering exhibits a stark contrast. Namely, $HO^{\phi}_{AC7} \geq H_{AC7}\phi \geq HO^{\phi}_{C_7} \geq H^{\phi}_{C_7}$. $HO^{\phi}_{C_7}$ has an efficiency about 12 percent above H_{C_7}, H_{AC7} is about 5.5 percent more efficient over HO_{C_7}, and HO_{AC7} is the most efficient, being about 13.2 percent above H_{AC7}.

Permuted Domain-Ordering

The plot of Figure 7.8 employing straight line variable ordering with permuted domains, exhibits a completely different behavior from that of Figure 7.6. Heuristic $V^{\phi}_{C_7}$ and V^{ϕ}_{AC7} ($VO^{\phi}_{C_7}$ and VO^{ϕ}_{AC7}) exhibit very similar efficiencies. The projection

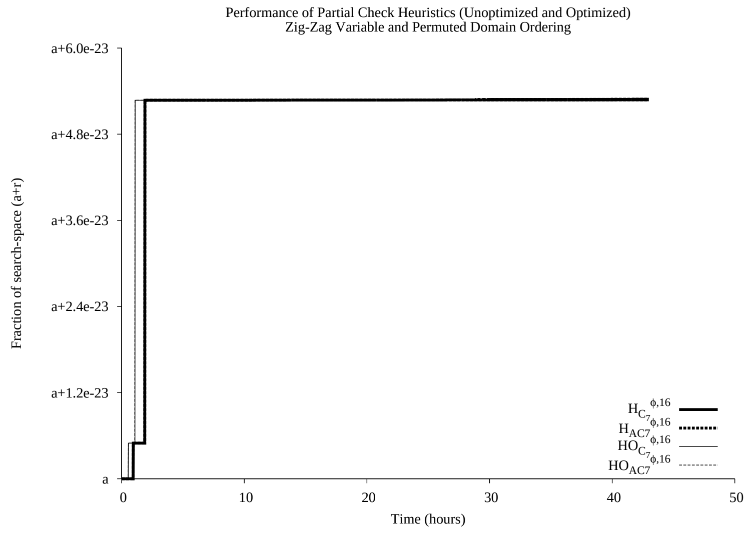


Figure 7.9: Performance of heuristics employing Zig-Zag Variable Ordering for permuted domains

of past assignments on to the domains of future variables, reducing these domains during the process, seem to have no effect. The optimization of skipping the arc-consistency check when a domain wipe-out occurs appears to yield a 31-percent efficiency over the unoptimized version.

The plot of Figure 7.9 using a zig-zag variable ordering with permuted domains exhibits a jump within the second minute of starting the experiment. At this point, a large number of assignments to variables has changed. Moreover, after the jump, the curve has remained almost constant, suggesting a large number of solutions with only the last few variable assignments changing while the first several values remained the same in this solution space.

7.8 Results on Symmetry

We report results on symmetry of DES S-boxes, and on the violations of one or more criteria due to non-simultaneous interchanges of rows, columns or quadrants. We also report the result of an experiment performed in an attempt to break symmetry by restricting domain-values.

Operation on DES S-box	**Total # of Configur-ations**	**# S-boxes, and impact on Score**	**Criteria violated By Remaining Configurations**
Interchange Rows (**S-4**)	32	16, no change	**S-4**
Interchange Columns (**S-5**)	2048	16, no change	**S-4**
Interchange Diagonals (**S-6**)	32	16, no change	**S-3**
Rotate S-box (**S-7**)	16	16, no change	None
Invert S-box entries	16	16, no change	None

Table 7.12: Observations made by interchanging Rows, Columns and Diagonals of all eight DES S-boxes

7.8.1 Symmetry of DES S-boxes

We have verified row, column, diagonal, rotational and bit inversion symmeties of all the eight S-boxes of DES. All configurations were considered, including those in which simultaneous interchanges were not done. Table 7.12 summarizes the results of the experiments on these configurations.

7.8.2 Breaking Symmetry by Restricting Domain-Values

Since each S-box possesses the property of Bit Inversion, there is a likelihood that restricting the domain of at least one variable will result in pruning of at least one future variable resulting in further reduction of the search space, and the consequent optimization. To verify this fact, we have performed the following experiment. Introduce a new variable x_{-1} that assumes a domain identical to that of the S-box variables. Add the following, new constraint between x_{-1} and x_0:

$$x_{-1} = 0 \wedge x_0 \in \{0, 1, 2, 3, 4, 5, 6, 7\}$$

The idea is to restrict the domain of x_0 to the set $\{0, 1, 2, 3, 4, 5, 6, 7\}$ without disturbing the symmetry of all existing constraints involving x_0. Upon running the solver now, the following was observed.

No further reduction has happened to the domain D_0 of x_0, suggesting that restricting the value(s) of any of the S-box variables to take advantage of the property of symmetry does not result in further optimization.

The examples of Chapter 6 suggest that once an S-box is found, a few others can be written down immediately due to Rotational and Bit Inversion symmetry properties of appropriate constraints. Unfortunately this idea will not work with `Solver`– an intelligent backtracker – and those symmetric S-boxes will end up

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	3	5	6	9	10	15	12	7	8	2	1	14	13	4	11
1	3	0	6	5	10	12	9	15	4	13	8	14	1	11	7	2
2	9	5	15	3	10	12	6	0	4	2	1	14	7	11	8	13
3	5	9	10	0	3	6	15	12	8	14	13	7	4	1	2	11

Figure 7.10: A 6×4 S-box S'_7 with score 8, generated by the Solver. This S-box is used in the place of DES S-box S_7.

eventually getting reproduced. We have not continued with further investigation of symmetry for the time-being.

7.9 Results on Criterion S-8

We have formulated **S-8** based on equation 6.24. This constraint works on $8! = 40320$ permutation of eight S-boxes taken all at a time. The difference-score given by equation 6.24 is compared in this section for various arrangements.

7.9.1 The DES S-boxes S_1 to S_8

For the existing arrangement $(S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8)$, the difference-score evaluates to 1120. There are 832 arrangements having this difference-score.

However, this is not the minimum difference-score. The minimum value is actually equal to 1024. There are a total of 256 such arrangements. An example arrangement is $(S_2, S_1, S_4, S_3, S_6, S_5, S_8, S_7)$.

7.9.2 Eight "best" S-boxes generated by the Solver

The minimum difference-score of the first eight S-boxes having a score equal to 8, generated by the solver, evaluates to 82944. We need to develop a heuristic to discard an S-box, substitute it with another, and attempt to minimize the difference-score.

7.9.3 Replacing a DES S-box

We have replaced DES S-box S_7 having the "worst" score equal to 18, with the S-box S'_7 of Figure 7.10 generated by the CSP approach having a score equal to 8.

The result is that the arrangement $(S_2, S_1, S_4, S_3, S_6, S_5, S_8, S'_7)$ yielded a difference-score equal to 960, a value "better" than that of the existing DES S-boxes with the same arrangement.

Chapter 8

Discussion

In this chapter, we summarize the observations made from the experiments in Chapter 7. We will also add pointers to more work that can be done in S-box generation using CSPs.

8.1 Small-size S-box Generation using CSPs

We are able to generate 4×2 and 5×3 S-boxes. The criteria specified in Chapter 3 do not simultaneously satisfy S-boxes of these sizes. Most criteria, particularly binary constraints **S-3** to **S-6**, and the n-ary constraint **S-7**, pose a requirement on the size of the inputs to an S-box, with the consequent outputs based on these requirements. Some criteria had to be relaxed. For a **S-4**2 S-box, criterion **S-7** does not apply while combinations of the other criteria **S-3** to **S-6** are considered in Table 7.3. For 5×3 S-boxes, criteria **S-5** and **S-6** have been relaxed. The reason is that, for criterion **S-5**, there are no "middle two" bits accessible in a 5-bit S-box input. As for **S-6**, implementing this check did not result in any solutions.

The smallest S-box that can possibly go through all eight criteria needs an input bitlength of 6, such as those used in DES.

8.2 Complete 6×4 S-box Generation

A complete S-box, which we ordinarily refer to as an S-box, is one in which all variables are assigned. Based upon the experiments detailed in Section 7.6 for 6×4 S-box generation starting from a thresholding score of $\tau = 16$, we have found that the complete, non-incremental heuristic $H_S^{\phi,16}$ indeed generate S-boxes but barely around 384 S-boxes over a two-day run. This gets speeded up by a factor of 100,000 when we formulated our idea of partially-assigned S-boxes. Heuristics $H_C^{\phi,16}$, $H_{C7}^{\phi,16}$ and $H_{AC7}^{\phi,16}$ are the resulting heuristics. However, with this specified threshold, these heuristics have not visited the search space that contained many S-boxes with better scores such as 14 and below.

Using an incomplete, incremental heuristic $H_I^{\phi,16}$, we are able to obtain S-boxes of better scores such as 14, 12, 10 and even 8. The highlight of this heuristic is its ability to yield S-boxes having scores better than the best S-boxes of DES having scores 10, namely, score 8. We have performed a proof-of-concept using a few of these S-boxes having score 8. In this trial, all eight S-boxes have been replaced by the few chosen S-boxes in an implementation of DES. Encryption and decryption works as expected.

We have run experiments on random-permutation-and-restarts of domains. None of the complete heuristics have yielded S-boxes having score equal to 8. At most, we have obtained S-boxes with scores up to 10. The incomplete heuristic has, however, yielded S-boxes with a score up to 8 using domain permutation.

Based on the experiments performed for generating completely filled S-boxes, one can conclude that the most promising heuristic for efficient exploration is $H_{AC7}^{\phi,\tau}$. However, the heuristic that yields the best quality of S-boxes as measured by Matsui's metric is $H_{I}^{\phi,\tau}$.

8.3 Performance Comparison of Complete, Incremental Heuristics

On the basis of the experiments in Chapter 7, one can safely conclude that regardless of optimization or variable ordering, the incremental heuristic with projection of past assignments on to future constraints performs more efficiently compared to the incremental heuristic that treats **S-7** as a single n-ary constraint. This is true of both, ascending-ordered domains and permuted domains given the permutation with seed 1000. For the zig-zag variable ordering heuristic, the percentage of efficiency goes up from about 5% to about 15% as reported in the experiments.

8.4 Effect of Variable and Value Ordering

We have considered two forms of variable ordering for 6×4 S-box generation: The default straight-line variable ordering in which each S-box is populated by assigning values in a row-wise fashion, and Zig-Zag variable ordering in which the entries are assigned in a zig-zag manner for the first two rows, followed by the last two rows.

Intuitively, we felt that straight-line variable ordering should have yielded better results compared to zig-zag ordering, in terms of performance. However, our results were surprising. The zig-zag pattern appeared to perform way better in comparison with the straightforward straight-line variable ordering. When domains were permuted, the results were even more interesting for zig-zag variable ordering. Within the first few minutes over a two-day run, the search jumped to the farthest point ever encountered in our systematic search (to a scale of 10^{-23} as the plot of Figure 7.9 suggests) and remained constant thereafter. This suggests that there are a large number of S-boxes with score of 16 and below encountered in this search space, given this permutation.

We have obtained a newer result. Heuristic $HO_{AC7}^{\phi,16}$ (zig-zag variable ordering) yields S-boxes having scores equal to 14, 12 and even 10 when a threshold of 16 is specified, in the permuted-domain-space. This has never happened when the domains are default-ordered, using any of the complete heuristics, given that the threshold is $\tau = 16$. Emboldened with this achievement, we have attempted to re-run the same experiment with thresholds $\tau = 14, 12$ to see if we can get S-boxes with scores 8 and below. So far, we have obtained S-boxes with scores up to 10. We are now running the experiment with thresholds $\tau = 10, 8$ to see what is happening.

In CSP literature, variable ordering has been suggested as a promising alternative for efficient search-space exploration. We can conclude that merely ordering variables in some fashion need not necessarily give very promising results, always. The straight-line heuristic is a case in point. The zig-zag heuristic appeared to be more promising. The fact that the plot of Figure 7.9 became constant after the jump suggests that there are a large number of S-boxes with score at most 16 in this search space. The heuristic worked for this permutation. For some other permutation, it may very well prove to be more inefficient. In our case, the search space is very large. The nature of the search space plays a role in deciding on the variable ordering heuristic.

A promising variable ordering heuristic we need to try with, is to select the next variable having least domain cardinality to make the next assignment, and continue further. This is well-known in CSP literature and it will be interesting to study the effect of this heuristic on our application.

As for value (domain) ordering, although we have employed random ordering (with a seed of 1000 in the experiments of Chapter 7), a probabilistic strategy to select the next value assigned to a variable can be used. The literature discusses a metric called *promise value* associated with a domain-value selected. The greater the promise value, the better. We can then permute the domains of variables in the descending order of promise values. Note that after the first few variables are assigned, the promise values of subsequent domains are likely to change, suggesting a *dynamic* permutation of the domains for variables, one by one as and when an assignment is made.

8.5 Effect of Optimization

The optimization of all heuristics, discussed in subsection 5.5.1, has yielded slight speed-up. For example, for a straight-line variable ordering, heuristic $VO_{C_7}^{\phi,\tau}$ is around 1.1 percent more efficient than $V^{\phi,\tau}_{C_7}$. Similar is the case of $VO^{\phi,\tau}_{AC7}$ over $V^{\phi,\tau}_{AC7}$. With zig-zag variable ordering, the optimization yielded slightly more promise (1.4% in both cases).

8.6 Symmetry

Symmetry has been addressed in S-boxes based upon the constraints that possess the property of symmetry. We have seen conditional row, column and diagonal symmetry that need not always yield S-boxes. Rotational and Bit Inversion Symmetry however, yields alternative S-boxes. Theoretically, due to these two forms of symmetry, a 400%-speedup is achieved due to the fact that (1) Bit Inversion Symmetry reduces the search space by half, and (2) Rotational Symmetry reduces further by half.

We have also experimented measuring the impact of symmetry on search, by adding a new symmetry-breaking constraint. The speedup encountered is insignificant.

Rejecting an S-box simply because it is symmetric (and its score did not change) need not necessarily work. The reason is, in criteria **S-8**, the probability P with the S-box and its symmetric version *may* be different when either S-box is brought into interaction with the other seven. Unless the contrary is proved, we will still need to have all S-boxes and their symmetric versions, which cannot be discarded. This needs to be investigated further and accordingly, consideration is made of whether to add new symmetry- breaking constraints to reduce the solution space.

Chapter 9
Conclusions

We conclude this Dissertation by summarizing our contributions and enumerating the limitations and future directions of our work on employing CSPs to generate S-boxes.

9.1 Summary of Contributions

We have proposed a novel approach to the design of S-boxes for Feistel Ciphers, an example of which is the Data Encryption Standard (DES). For the purposes of this Dissertation, the eight security criteria of DES have been formulated into constraints. These eight criteria are numbered **S-1** to **S-8**.

For CSP formulation, variables have been identified along with their domains. **S-1** is already inherent in the choice of variables and has not been discussed further. **S-8** deals with multiple S-boxes and is discussed separately. **S-3** to **S-6** are binary constraints which have been precompiled into a solver `Solver` that outputs solutions to satisfy these constraints. **S-7** and **S-2** are n-ary global constraints which have been formulated as heuristics to run on the solutions to generate the S-boxes. The finer aspects of our contributions are now presented.

9.1.1 Heuristics

Non-incremental Heuristic $H_S^{\phi,\tau}$ This heuristic checks to see if **S-2** and **S-7** is satisfied. This is a naiv̈e implementation employing systematic generate-and-test, and is very inefficient.

Incremental, Incomplete Heuristic $H_I^{\phi,\tau}$ This heuristic checks after each assignment to see if **S-2** is satisfied, with **S-7** implemented as an n-ary global constraint. This heuristic has been the most promising among all for generating high- quality S-boxes, having generated S-boxes with a score of 8, superior to the best score of all of the eight S-boxes of DES. The two figures below display two such S-boxes, the first being with a score of 8 and the one following it, with a score equal to 10.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	3	5	6	9	10	15	12	7	4	14	13	2	1	8	11
1	3	0	6	5	10	9	12	15	4	7	13	14	1	2	11	8
2	3	15	0	12	5	6	9	10	4	8	7	11	14	13	2	1
3	0	12	3	15	9	10	5	6	7	11	4	8	2	1	14	13

Figure 9.1: A 6×4 S-box with score 8, generated by our solver

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	3	5	6	9	10	15	12	7	4	14	13	2	1	8	11
1	3	0	6	5	10	9	12	15	4	7	13	14	1	2	11	8
2	3	15	0	12	5	6	9	10	4	8	11	2	14	13	7	1
3	0	12	3	15	9	10	5	6	7	11	4	8	2	1	14	13

Figure 9.2: A 6×4 S-box with score 10, generated by our solver

Incremental, Complete Heuristic $H_C^{\phi;\tau}$ We have formulated the notion of a *partially assigned* S-box. The following are new properties of linear approximation tables for partially assigned S-boxes.

1. For any a,b,X',Φ', $0 \leq N_{X'}^{\Phi'}(a,b) \leq |X'|$.
2. For any a,b,u,X',Φ', $N_{X' \cup \{u\}}^{\Phi'}(a,b) - N_{X'}^{\Phi'}(a,b) \in \{0,1\}$.

The condition on when a partially assigned S-box can be extended to a full S-box that will satisfy **S-2**, is:

$$|X'| - \tau - \frac{|X|}{2} \leq \max_{a,b} N_{X'}^{\Phi'}(a,b) \leq \frac{|X|}{2} + \tau$$

S-7 is implemented as an n-ary global constraint.

Incremental, Complete Heuristic $H_{C_7}^{\phi;\tau}$ Having provided for incrementally checking to see if **S-2** is satisfied, the next logical step is to incrementalize the check for **S-7**.

Incremental, Complete Heuristic $H_{AC7}^{\phi;\tau}$ The constraint for criterion **S-7** – the COUNT constraint – is a global n-ary constraint that cannot be straightforwardly decomposed into binary constraints. Nevertheless, we have formulated a novel heuristic by which past assignments of values to variables are projected onto domains of future variables, reducing the domains during the process and improving upon efficiency.

We have proved that $H_{AC7}^{\phi,\tau}$ and $H_{C_7}^{\phi,\tau}$ produce identical sequences of S-boxes.

9.1.2 Optimization to `Solver`

For the complete heuristics $H_S^{\phi,\tau}$, $H_{C_7}^{\phi,\tau}$, and $H_{AC7}^{\phi,\tau}$, we have introduced an optimization into `Solver` by not checking for arc-consistency if the deletion-set is empty. The resulting heuristics are $HO_S^{\phi,\tau}$, $HO_{C_7}^{\phi,\tau}$, and $HO_{AC7}^{\phi,\tau}$, respectively. The performance of the optimized heuristics is marginally higher than that of the corresponding, unoptimized heuristics.

9.1.3 Order of Efficiency resulting from Ordering of Variables

We have found that visiting the variables in a zig-zag fashion generates S-boxes more efficiently compared to the straight-line manner in which the variables are visited over each row of an S-box. We can also conclude that variable-ordering, although promising in most cases, should account for the nature of search space to see if it has many solutions for efficient S-box generation, particularly for large-sized search spaces.

9.1.4 Order of Efficiency resulting from Ordering of Domains

We have found that the efficiency of `Solver` is much higher (about 35%) for permuted domains when using a zig-zag variable ordering, compared to straight-line variable ordering. We are also able to uncover a large number of S-boxes having lesser scores. With the threshold $\tau = 16$, S-boxes with scores equal to 16, 14, 12 and 10 have been uncovered. This has not been the case with the default (ascending order) domain-ordering where only $H_I^{\phi,\tau}$ emitted these lower-score S-boxes.

9.1.5 Search Efficiency Metric

For addressing performance of heuristics on search spaces using systematic search, merely counting the number of S-boxes is not sufficient. To arrive at a better understanding of the search space, we have formulated a metric to measure search efficiency by defining the fraction of search space traversed by each of our heuristics. Heuristic $H_{AC7}^{\phi,\tau}$ using zig-zag variable ordering on randomly-permuted domains offers traversal of a large amount of search space in the shortest possible time compared to the others, based on this metric.

Constraint for Criterion	**Row Symmetry**	**Column Symmetry**	**Diagonal Symmetry**	**Rotational Symmetry**	**Bit Inversion Symmetry**
S-2					×
S-3					×
S-4	×				×
S-5		×			×
S-6			×		×
S-7				×	×

Table 9.1: Summary of Results on Symmetry of constraints modeling S-box Criteria

9.1.6 New Forms of Symmetries

Our CSP methodology has exposed new forms of symmetry in S-boxes and these are summarized in the table below.

Rotational and bit inversion symmetry are the only two forms of symmetry that result in an S-box that satisfies all constraints. We have proved the invariance of the score of an S-box over these two symmetries. Symmetry is broken by adding more constraints.

9.2 Limitations and Future Work

This Dissertation attempts to model the eight criteria specified in [16] as a CSP. The limitations of our formulation, heuristics, symmetry, variable and value ordering will now be addressed along with pointers in future directions.

9.2.1 `Alldiff` *Constraints*

S-box criteria **S-3** to **S-4** result in binary constraints in which the values to two variables are unequal. All of these are in the category of the `Alldiff` constraint in various ways. A future task is to model all of these constraints into `Alldiff` constraints, and employ efficient algorithms to process this special form of constraints, using an efficient algorithm by Puget [49].

9.2.2 Variable Ordering

A form of variable ordering prescribed in CSP literature is to select the next variable for assignment, that has the least domain cardinality. It is interesting to study the performance of this variable ordering heuristic against straight line and zig-zag ordering heuristics used in our application.

Another form of *dynamic* variable ordering is worth exploring. We can list the variables in the descending order of the number of times they appear in the constraints. We expect around half the number of variables (32 out of the 64, for 6×4 S-boxes) in this listing. The next variable to be selected for assignment will then be among the most-constrained variables.

The selection of the next variable from the variable-ordering need not be made in the beginning. In fact, as equation 4.9 suggests, at least $\frac{|X|}{2}$ variables should be assigned before partial checks are made for **S-2**. And for **S-7**, as discussed at the end of section 5.3 and throughout section 5.4, at least $\frac{|X|}{2} + 8$ variables should be assigned before initiating any partial checks. One can empirically select the next variable from this point onwards. In general, one should parameterize the point of selection by a parameter α such that the variable x_α should be the next variable from where selection should begin based upon the variable-ordering. For example, when $\alpha = 0$, begin selecting from the first variable in the variable-ordering. When $\alpha = \frac{|X|}{2}$, begin this selection from the middle, and so on. One may choose α to be below, or above, the middle variable (for example, $\alpha = \frac{|X|}{2} + 8$) and see how the asymmetry helps. Alternatively, examine the performance of the solver by sequentially varying α, and determine empirically the optimum point for α for the particular heuristic. This should be done for ordered as well as permuted domains.

9.2.3 Value Ordering

Instead of randomly shuffling the domains of each variable, assign, based upon the current state of the S-box, a promise value to each of the (reduced) domain-elements for the current variable being assigned. The higher the promise value, the better. Next, select the domain-element with the highest promise value and assign to the variable. The promise values for the subsequent domains may now change. Repeat the process.

9.2.4 Symmetry

Prove or disprove that even after performing rotational and bit inversion transformations to obtain symmetric S-boxes whose scores do not change, the probability P of Equation 6.24 does not change. If P does not indeed change, then symmetry can be broken by adding new constraints and symmetric S-boxes can be discarded, reducing the search space and improving upon efficiency.

9.2.5 The Score $\sigma_X(\Phi)$ of the S-box Φ

We have been able to experimentally obtain 6×4 S-boxes with the “best” score equal to 8. Using our solvers, we have been unable to go below this value in the time-frame specified in our experiments. Is it actually possible to obtain S-boxes with better scores (values of 6, 4, 2 and ideally, 0)? Do such S-boxes even exist?

We don't know. If we are able to use information-theoretic results to *prove* (or disprove) the existence of 6×4 S-boxes with such lesser scores, or even determine the minimum value of the score, it will be a key result.

In section 3.3, we mentioned two approaches to projections. In one, assignment to the current variable is projected on past assignments. This approach is employed for n-ary constraints **S-2** (section 4.4) and **S-7** (section 5.3). The problem with this approach is that there are still partial *checks* that need to be carried out. For **S-2**, the score threshold τ is built into these partial checks (equations 4.9 and 4.12). The second approach is of projecting domains of future variables onto past assignments, which is discussed for **S-7** (section 5.4). The advantage of this approach is that after domains of future variables are reduced, no explicit checks are required, and the next assignment always results in an S-box. If we are able to employ this approach for the nonlinear constraint **S-2** with τ as a parameter, we can generate S-boxes with scores of 8, 6, 4, 2 and even 0 in real-time! In particular, if no such S-boxes with scores equal of 6, 4, 2 or 0 are found, the solution space would be empty and we would have proved the result experimentally – a very important result.

9.2.6 Adding New Security Criteria

There have been more advances in work on S-box design during and after publication of the eight design criteria. Some examples are the use of Bent functions in S-box design, avalanche properties and strict avalanche criteria, bit independence criteria and higher-order bit independence criteria. These can be modeled into additional constraints and input to the CSP.

9.2.7 Almost-Similar S-boxes

In all of our complete search heuristics, the first several S-boxes generated have always been possessing identical rows and columns. Although this is not an issue of "symmetry", it is an issue of "similarity" of S-boxes. If we have to select 8 S-boxes and arrange them to satisfy **S-8**, we want to ensure that they should never possess identical rows and columns. For this, we need to come up with a measure to remove "similar", or "almost-similar" S-boxes. This is another direction along which we would like to proceed further.

9.2.8 Systematic Sampling of Performance Measures

We have formulated equation 7.6 for the measure of performance and have compared heuristics by comparing the values of $(a + p)$ in this equation. To determine a percentage, we have taken those points on the performance curves when the "jumps" are encountered in these curves. The reason we considered those "jumps" is that this was where we could easily read off the variations in heuristics and calculate percentage differences. We could have very well chosen (consistently) a different point on the performance curves. Instead of following an ad-hoc approach and sampling arbitrary points, a more systematic sampling could be done as follows. Measure the area under the performance curves and determine their differences up to a specified time (for example, at the end of five hours). The changes in the areas will represent the necessary speed-ups.

9.2.9 Other Solvers

As mentioned in Section 2.6.2, Mozart-Oz is a programming language used for constraint programming. We attempted to model our S-box problem as a CSP using this programming language, but quickly discovered that it does not lend itself flexibly for bit-level operations. Commercial solvers such as ILOG^{TM} have not been evaluated due to budget requirements. Not only that, customizing and tailoring the solver to suit our requirement is a grey area and purchasing such software involves making a feasibility analysis, something that is to be done at a different level. We have instead employed a home-grown solver implemented in C^{++} for our purposes. Formulating this problem as a CSP using other solvers will be considered eventually, particularly for the purposes of implementing new security criteria.

9.2.10 NP-Completeness

Our main objective is to maximize nonlinearity by minimizing the score $\sigma_X(\Phi)$ for an S-box Φ. Another measure, not specified in the list of criteria of Table 2.1 and not modeled in our work, is *autocorrelation* [15, 41]. The smaller the measure of autocorrelation, the better. Designing an S-box that has maximum nonlinearity and minimum autocorrelation is known to be NP-complete [41]. In general, solving a CSP is also known to be NP-complete. Can we conclude that the S-box Design Problem is NP-complete solely on the basis of these arguments? To actually prove this result, one needs to first prove that the given problem is in NP. Next, find a problem known to be NP-complete and employ a construction to transform this known problem to the equivalent S-box design problem.

9.3 An Alternative CSP Based Approach to Model all Eight Criteria

We have modeled criteria **S-2** to **S-7**. Criterion **S-8** could not be modeled by this framework alone. An explicit check of criterion **S-8** after generation of eight S-boxes had to be done.

A way to ensure that the *entire* set of criteria **S-2** to **S-8** is modeled strictly as a CSP[1] is the following.

Instead of formulating 64 variables for one 6×4 S-box, formulate $64 \times 8 = 512$ variables for eight 6×4 S-boxes. In other words, the variables $X = \{x_0, x_1, \ldots, x_{63}, x_{64}, \ldots, x_{511}\}$. The domains are identical for all of these variables in X, equal to $\{0, 1, \ldots, 15\}$. A solution to this problem is an assignment to all 512 variables. In other words, the solution generates *eight* S-boxes S_i where each S-box S_i has variables $\{x_{64i}, x_{64i+1}, x_{64i+2}, \ldots, x_{64i+63}\}$, $0 \leq i < 8$.

9.3.1 The Formulation of Constraints for Individual S-boxes

The constraints for **S-2** to **S-7**, governed by equations 4.7 to 5.1 will now be identical for the variables in S-box S_i, $0 \leq i < 8$. Since these criteria gives rise to 672 binary constraints for criteria **S-3** to **S-6** for each S-box S_i, the total number of binary constraints in this formulation will equal $672 \times 8 = 5376$.

9.3.2 Modeling Criterion **S-8**

Criterion **S-8** is now modeled as an n-ary constraint as follows (refer section 6.10). Let

$$\begin{aligned} Q_{0,i} &= \max\{D_i(3,0), D_i(7,0), D_i(11,0), D_i(15,0)\} \\ Q_{1,i} &= \max\{D_i(50,0), D_i(54,0), D_i(58,0), D_i(62,0)\} \\ Q_{2,i} &= \max\{D_i(32,0), D_i(36,0), D_i(40,0), D_i(44,0)\} \end{aligned}$$

where $D_i(a,b)$ is the entry in the XOR table for S-box S_i under row a, column b. Arrange the eight S-boxes S_i, $0 \leq i < 8$, so as to minimize the following probability:

$$P = \max_{i=0,2,\ldots,7} Q_{0,i \bmod 8} \cdot Q_{1,(i+1) \bmod 8} \cdot Q_{2,(i+2) \bmod 8} \tag{9.1}$$

We note the difference in the way the modulus is taken, in this equation as compared to equation 6.24. There, $1 \leq j \leq 8$ while in equation 9.1, $0 \leq i < 8$.

[1] This idea is due to Dr. Philip Chan.

The main advantage of this approach to modeling the S-box problem as a pure CSP approach is that pruning can now occur *across* S-boxes due to the constraint 9.1 for criterion **S-8**. The limitation of this approach is that the search space has now increased from 16^{64} for 6×4 S-boxes, to 16^{512}, i.e. by a factor of 16^{8} which is in itself exponential. We need to formulate novel heuristics for **S-8** to project domains and generate solutions. This is in addition to the formulations of projection-based heuristics for criteria **S-2** and **S-7** discussed, respectively, in chapters 4 and 5.

9.3.3 Experimental Observations and Issues

We have modeled the S-box problem for 6×4 S-boxes and have formulated constraints involving all 512 variables. The variable ordering selected by us is the zig-zag ordering and domains are ordered in ascending order. The nonlinearity threshold (for **S-2**) is chosen equal to 16. The following observations are made.

S-box Generation In the first solution, eight S-boxes are identical. This is to be expected due to the systematic nature of the search.

After an S-box set of eight S-boxes is generated, the last few variables (around 30) of only the *last* S-box change values for the next solution that forms the next set of eight S-boxes. The remaining seven S-boxes out of these eight have not changed in their entries in our experiments thus far. This is expected due to the exponential nature of the search space.

Difference Score equal to zero The difference-score of all of the eight S-boxes in the first solution, and in the next few, turn out to be *zero*! Does this mean that this S-box set is better in comparison with what we have found thus far? Cryptanalytically it does not appear to be so.

To check whether eight identical S-boxes always yields a difference-score equal to zero or not, we have run a check for **S-8** on eight identical copies of DES S-box **S-1**. The difference-score evaluates to 1008, not zero.

We need to interpret the difference-score and possibly threshold the same. The configuration of eight identical S-boxes generated by us, and many others having the first several identical S-boxes, may have to be rejected based on the threshold.

We can use a maximum threshold to limit search. Since we know that the best score (Section 7.9.3) is 960, this can be used as the upper threshold to reject S-box sets that yield a higher difference-score.

9.3.4 Improvements – Search Speedup

Using this alternative model, revisiting of S-boxes in the search-space need to be avoided. Also, there are two levels in which search can be speeded up. These considerations are discussed.

Revisiting of Solutions Criterion **S-8** works on the 8! permutations of a set of eight S-boxes. These 8! permutations are visited at the end of every complete assignment to all 512 variables.

A way to speed up is to *mark* or *encode* each permutation. We see that each permutation will eventually get revisited as part of the systematic search. When this happens, examine if they have been marked before and if so, discard the solution.

Different Sets of S-boxes In this idea, we attempt to exercise a check on *sets* of eight S-boxes. Let the S-boxes be labeled as $S_1, S_2, \ldots, S_8$. The eight S-boxes may appear in different orders at several search-points. For example, a complete assignment may eventually yield an S-box set that corresponds to a permutation of the eight S-boxes just labeled.

We do not have to determine the quantity governed by Eq. 9.1 for all 8! permutations *after each complete assignment*. Instead, it is enough to determine this quantity only for the eight triplets $S_j, S_{j \bmod 8+1}, S_{(j \bmod 8+1) \bmod 8+1}$ where $1 \leq j \leq 8$, since each permutation gets visited as part of the search.

9.4 Concluding Remarks

We have addressed the "age-old problem of S-box design" using CSP methodology. This is a known, hard problem, which we have visited using the novel approach of systematic search using CSPs. During the process, we have obtained S-boxes with superior quality metric compared to those of the best DES S-boxes. We have applied the properties of CSPs to formulate heuristics, and have derived new results not known in the literature on S-box design. New S-box symmetries have been discovered in this work. For the purpose of systematic search, a novel quantification of search efficiency has been proposed.

We are not claiming to have solved the S-box design problem, but have discovered new results in S-box design using a modeling technique that provides avenues for expansion into future research on S-box design.

References

[1] Data encryption standard (DES). Federal Information Processing Standard (FIPS) 46-2, January 1988.

[2] Advanced encryption standard (des). Federal Information Processing Standard (FIPS) 197, November 2001. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[3] Digital signature standard (DSS). Federal Information Processing Standard (FIPS) 186-3, June 2009.

[4] Carlisle Adams. Cast-256 algorithm specification.

[5] Carlisle Adams and Stafford Tavares. The structured design of cryptographically good s-boxes. *J. Cryptol.*, 3(1):27–41, 1990.

[6] Carlisle M. Adams and Stafford E. Tavares. Good S-boxes are easy to find. In *CRYPTO '89: Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, pages 612–615, London, UK, 1990. Springer-Verlag.

[7] C.M. Adams and S.E. Tavares. Generating and counting binary bent sequences. *IEEE Transactions on Information Theory*, 36(5):1170–1173, Sep 1990.

[8] Giampaolo Bella and Stefano Bistarelli. Soft constraints for security protocol analysis: Confidentiality. In *Practical Aspects of Declarative Languages, Proceedings of the Third International Symposium on Practical Aspects of Declarative Languages (PADL 2001), Las Vegas, Nevada, March 11-12*, pages 108–122, 2001.

[9] Christian Bessière and Jean-Charles Régin. Refining the basic constraint propagation algorithm. In Bernhard Nebel, editor, *IJCAI*, pages 309–315. Morgan Kaufmann, 2001.

[10] Christian Bessière, Jean-Charles Régin, Roland H. C. Yap, and Yuanlin Zhang. An optimal coarse-grained arc consistency algorithm. *Artif. Intell.*, 165(2):165–185, 2005.

[11] Eli Biham and Adi Shamir. *Differential cryptanalysis of the data encryption standard*. Springer-Verlag, London, UK, 1993.

[12] G.R Blakley. Safeguarding cryptographic keys. In *Proc. 1979, National Comp. Conf.*, volume 48, pages 313–317, New York, June 1979. American Federation of Information Processing Societies Press.

[13] A. Chakraborty, A. Chatterjee, and S.K. Basu. Performance of various cost functions in the search for strong s-boxes. *Information Technology, 2006. ICIT '06. 9th International Conference on*, pages 188–189, Dec. 2006.

[14] J.A. Clark, J.L. Jacob, S. Maitra, and P. Stanica. Almost boolean functions: the design of boolean functions by spectral inversion. *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, 3:2173–2180 Vol.3, Dec. 2003.

[15] J.A. Clark, J.L. Jacob, and S. Stepney. The design of S-boxes by simulated annealing. *Evolutionary Computation, 2004. CEC2004. Congress on*, 2:1533–1537 Vol.2, June 2004.

[16] D. Coppersmith. The data encryption standard (des) and its strength against attacks. *IBM J. Res. Dev.*, 38(3):243–250, 1994.

[17] Nicolas T. Courtois, Guilhem Castagnos, and Louis Goubin. What do DES S-boxes say to each other? Cryptology ePrint Archive, Report 2003/184, 2003. `http://eprint.iacr.org/`.

[18] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael, September 1999. `http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf`.

[19] Ivan Damgård, Matthias Fitzi, Jesper Buus Nielsen, and Tomas Toft. How to split a shared secret into shared bits in constant-round. cryptology eprint archive, report 2005/140, 2005.

[20] Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, October 2003.

[21] Horst Feistel. Cryptography and computer privacy. 228:15–23, 1973.

[22] Matthew L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[23] Kishan Chand Gupta and Palash Sarkar. Construction of high degree resilient S-boxes with improved nonlinearity. *Inf. Process. Lett.*, 95(3):413–417, 2005.

[24] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In Chris S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95); Vol. 1*, pages 607–615, Montréal, Québec, Canada, August 20-25 1995. Morgan Kaufmann, 1995.

[25] Thomas H.Cormen, Charles E.Leicerson, Ronald L.Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*, page 103. The MIT Press, Cambridge, Massachusetts, 2003.

[26] Howard M. Heys. A tutorial on linear and differential cryptanalysis. *Cryptologia*, XXVI(3):189–221, 2002.

[27] W.J. Van Hoeve. The alldifferent constraint: A survey. In *In Proceedings of the Sixth Annual Workshop of the ERCIM Working Group on Constraints*, 2001.

[28] Don Johnson and Alfred Menezes. The elliptic curve digital signature algorithm (ecdsa). Technical report, 1999.

[29] Grzegorz Kondrak and Peter van Beek. A theoretical evaluation of selected backtracking algorithms. In *IJCAI*, pages 541–547, 1995.

[30] Richard E. Korf. Improved limited discrepancy search. In *AAAI/IAAI, Vol. 1*, pages 286–291, 1996.

[31] Piotr Kotlarz and Zbigniew Kotulski. On application of neural networks for s-boxes design. In *AWIC*, pages 243–248, 2005.

[32] Xuejia Lai and James L. Massey. A proposal for a new block encryption standard. In *Advances in cryptology - EUROCRYPT '90*, pages 389–404. Springer-Verlag New York, Inc., 1991.

[33] Susan Landau. Communications security for the Twenty-first Century: The Advanced Encryption Standard. 47(4):450–459, April 2000.

[34] Mitsuru Matsui. Linear cryptanalysis method for des cipher. In *EUROCRYPT '93: Workshop on the theory and application of cryptographic techniques on Advances in cryptology*, pages 386–397, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[35] R. J. McEliece. A Public-Key Cryptosystem Based On Algebraic Coding Theory. *Deep Space Network Progress Report*, 44:114–116, January 1978.

[36] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC, http://www.cacr.math.uwaterloo.ca/hac/, October 1996. `http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\&amp;path=ASIN/0849385237`.

[37] William Millan. How to improve the nonlinearity of bijective s-boxes. In *ACISP '98: Proceedings of the Third Australasian Conference on Information Security and Privacy*, pages 181–192, London, UK, 1998. Springer-Verlag.

[38] S. Mister and C. Adams. Practical S-Box design, 1996.

[39] Ugo Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Inf. Sci.*, 7:95–132, 1974.

[40] M.Robshaw. Block ciphers. 1995. `http://www.rsasecurity.com/rsalabs/index.htm`.

[41] Nadia Nedjah and Luiza de Macedo Mourelle. Designing substitution boxes for secure ciphers. *Int. J. Innov. Comput. Appl.*, 1(1):86–91, 2007.

[42] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.

[43] Luke O'Connor. An analysis of a class of algorithms for s-box construction. *J. Cryptology*, 7(3):133–151, 1994.

[44] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[45] J. Pieprzyk and G. Finkelstein. Towards effective nonlinear cryptosystem design. *Computers and Digital Techniques, IEE Proceedings -*, 135(6):325–335, Nov 1988.

[46] Bart Preneel, editor. *Properties of Linear Approximation Tables*, volume 1008 of *Lecture Notes in Computer Science*. Springer, 1995.

[47] Patrick Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9:268–299, 1993.

[48] Jean-Francois Puget. On the satisfiability of symmetrical constrained satisfaction problems. In Henryk Jan Komorowski and Zbigniew W. Ras, editors, *ISMIS*, volume 689 of *Lecture Notes in Computer Science*, pages 350–361. Springer, 1993.

[49] Jean-Francois Puget. A fast algorithm for the bound consistency of alldiff constraints. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/ Innovative applications of artificial intelligence*, pages 359–366, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[50] Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In *AAAI '94: Proceedings of the twelfth national conference on Artificial intelligence (vol. 1)*, pages 362–367, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence.

[51] Jean-Charles Régin. The symmetric alldiff constraint. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 420–425, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[52] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to share a secret. *Communications of the ACM*, 22(22):612–613, 1979.

[53] O. S. Rothaus. On "bent" functions. *J. Comb. Theory, Ser. A*, 20(3):300–305, 1976.

[54] Stuart J. Russell and Peter Norvig. *Artificial Intelligence — A Modern Approach*. Pearson Education, Inc., http://www.cacr.math.uwaterloo.ca/hac/, 2003.

[55] Daniel Sabin and Eugene C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *PPCP '94: Proceedings of the Second International Workshop on Principles and Practice of Constraint Programming*, pages 10–20, London, UK, 1994. Springer-Verlag.

[56] Daniel Sabin and Eugene C. Freuder. Understanding and improving the mac algorithm. In *CP*, pages 167–181, 1997.

[57] Bruce Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). In *Fast Software Encryption, Cambridge Security Workshop*, pages 191–204, London, UK, 1994. Springer-Verlag.

[58] Bruce Schneier. Applied cryptography — protocols, algorithms, and source code in c. Textbook, Chapter 12, pp.265-301, 2002.

[59] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. *The Twofish encryption algorithm: a 128-bit block cipher.* John Wiley & Sons, Inc., New York, NY, USA, 1999.

[60] Jennifer Seberry, Xian-Mo Zhang, and Yuliang Zheng. Systematic generation of cryptographically robust s-boxes. In *ACM Conference on Computer and Communications Security*, pages 171–182, 1993. `http://citeseer.ist.psu.edu/seberry96systematic.html`.

[61] Markus Stadler. Publicly veri able secret sharing. In *In Advances in Cryptology EUROCRYPT '96*, pages 190–199. Springer Verlag, 1996.

[62] William Stallings. Cryptography and network security - principles and practices. Textbook, Chapter 3, pp.86-90, 2003. `http://www.prenhall.com/stallings`.

[63] Miroslaw Szaban and Franciszek Seredynski. Designing cryptographically strong s-boxes with the use of cellular automata. *Ann. UMCS, Inf.*, 8(2):27–41, 2008.

[64] Stafford E. Tavares and Howard M. Heys. Avalanche characteristics of substitution-permutation encryption networks. *IEEE Trans. Comput.*, 44(9):1131–1139, 1995.

[65] Wade Trappe and Lawrence C. Washington. *Introduction to Cryptography with Coding Theory.* Prentice Hall, January 2002.

[66] M.R.C. van Dongen. AC-3$_d$ an efficient arc-consistency algorithm with a low space-complexity. In P. Van Hentenryck, editor, *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP 2002)*, volume 2470 of *Lecture notes in Computer Science*, pages

755–760. Springer, 2002. Two embarassing errors on the first page have been fixed:).

[67] Peter van Roy and Seif Haridi. *Concepts, Techniques, and Models of Computer Programming*. The MIT Press, March 2004.

[68] Richard J. Wallace. Directed arc consistency preprocessing. In *Constraint Processing, Selected Papers*, pages 121–137, London, UK, 1995. Springer-Verlag.

[69] Toby Walsh. General symmetry breaking constraints. In Frédéric Benhamou, editor, *CP*, volume 4204 of *Lecture Notes in Computer Science*, pages 650–664. Springer, 2006.

[70] Toby Walsh. Breaking value symmetry. In Dieter Fox and Carla P. Gomes, editors, *AAAI*, pages 1585–1588. AAAI Press, 2008.

[71] David Waltz. Understanding line drawings of scenes with shadows. In *The Psychology of Computer Vision*, pages 19–91. McGraw-Hill, 1975.

[72] A.F. Webster and S.E. Tavares. On the design of s-boxes. pages 523–534. Springer-Verlag, 1986.

[73] Xun Yi, Shi Xin Cheng, Xiao Hu You, and Kwok Yan Lam. A method for obtaining cryptographically strong 8×8 S-boxes. In *Global Telecommunications Conference, 1997. GLOBECOM '97., IEEE*, volume 2, pages 689–693, Nov 1997.