Algorithms for Rotation Symmetric Boolean Functions

Article · September 2012

CITATIONS
0

READS
79

4 authors, including:

Satrajit Ghosh
Acharya Pradulla Chandra College, KOLKATA, I…
13 PUBLICATIONS **2** CITATIONS
SEE PROFILE

Parthasarathi Dasgupta
Indian Institute of Management Calcutta
92 PUBLICATIONS **238** CITATIONS
SEE PROFILE

INDIAN INSTITUTE OF MANAGEMENT CALCUTTA

WORKING PAPER SERIES

WPS No. 713/ September 2012

Algorithms for Rotation Symmetric Boolean Functions

by

Subrata Das
Assistant Professor, Department of Information Technology,
Academy of Technology, West Bengal

Satrajit Ghosh
Assistant Professor, Department of Computer Science,
APC College, West Bengal

Parthasarathi Dasgupta
Professor, IIM Calcutta, Diamond Harbour Road, Joka, Kolkata 700104, India

&

Samar Sensarma
Professor, Department of Computer Science & Engineering University of Calcutta

Algorithms for Rotation Symmetric Boolean Functions

Subrata Das[1], Satrajit Ghosh[2], Parthasarathi Dasgupta[3], and Samar Sensarma[4]

[1] Department of Information Technology
Academy of Technology
dsubrata.mt@gmail.com
[2] Department of Computer Science
APC College
gsatrajit@gmail.com
[3] Management Information Systems Group
Indian Institute of Management Calcutta
partha@iimcal.ac.in
[4] Department of Computer Science & Engineering
University of Calcutta
sssarma2010@gmail.com

Abstract. Rotation Symmetric Boolean functions (*RSBF*) are of immense importance as building blocks of cryptosystems. This class of Boolean functions are invariant under circular translation of indices. It is known that, for n-variable *RSBF* functions, the associated set of input bit strings can be divided into a number of subsets (called *partitions* or *orbits*), where every element of a subset can be obtained by simply rotating the string of bits of some other element of the same subset. In this paper we propose algorithms for the generation of these partitions of $RSBF$s and implement for up to 26 variables.

Keywords: Algorithms,cryptography, Symmetric boolean functions,Rotation symmetric boolean functions

1 Introduction

A Boolean function $f^n(x_{n-1}, x_{n-2}, ..., x_0)$ of n variables is a mapping from F_2^n to F_2,where F_2^n is a n-dimensional vector space over the two-element field F_2. A Boolean function is symmetric if and only if it is invariant under any permutation of its variables [1]. Rotation Symmetric Boolean Functions(*RSBFs)* are a special class of Boolean function for which the function value will be same for any rotation of its variables [12]. For secret key cryptosystems, balancedness,nonlinearity,correlation immunity,algebraic degree [11] are the different criteria for choosing a Boolean Function for cryptographic applications. *RSBFs* have good combination of these properties [6],[4]. It is clear to see that there are 2^{2^n} boolean functions on n variables, and thus, searching for all these functions

exhaustively is exponential.Thus, in order to look for *RSBFs*, it is imperative to have an idea about the number of orbits(i.e. partitions) in rotation symmetric functions.

In this paper, we propose three simple algorithms for generating $RSBF$s of a given number of variables and implement upto 26 variables.

Rest of the paper is organized as follows.Section 2 reviews some related recent works. Section 3 introduces some terminologies to be used in subsequent discussions and Section 4 proposes three algorithms A,B and C for generating $RSBF$s of n variables. Section 5 briefly discusses the implementation of the algorithms A,B and C. Finally, Section 6 concludes the chapter and briefly states the future scopes of work.

2 Literature Review

An extensive study of symmetric Boolean functions, especially of their cryptographic properties has been done in [8]. In [2] Pieprzyk and Qu have studied a special type of symmetric Boolean functions, called rotation symmetric Boolean function.In that paper the authors have suggested that *RSBF* can be applied as round functions of a hashing algorithm such as MD5. Rotation symmetric Boolean functions (*RSBF*) are of great research interest for theoreticians as well as for practitioners in the field of cryptography [3]. Any symmetric boolean function (w.r.t. any permutation) is also rotation symmetric, but the converse is not always true [2]. In [4] the authors discuss some new results on *rotation symmetric correlation immune(CI) and bent* functions and important data structures for efficient search strategy of these functions. They also proved the non existence of the homogeneous rotation symmetric bent functions of degree ≥ 3 on a single cycle. In [5] the authors have proposed an efficient implementation of search strategy for rotation symmetric boolean function. In [6] theoretical construction of rotation symmetric boolean functions on odd number of variables with maximum possible algebraic immunity are discussed.The investigation of balanced *RSBF*s and 1^{st} order correlation immune *RSBF*s and enumeration formula for n-variable balanced *RSBF*s where n is a power of prime reported in [7].

3 Preliminaries

A of n variables Boolean function $f(x_{n-1}, x_{n-2}, ..., x_0)$ is said to be *Rotation Symmetric* if $f(x_{n-1}, x_{n-2},\ldots,x_0)$=$f(x_{n-2}, x_{n-2},\ldots,x_0,x_{n-1})$=....=$f(x_0, x_1,\ldots,x_{n-1})$. Thus, a *Rotation Symmetric Boolean function* remains invariant under cyclic permutation of its variables. The cyclic permutation may be in either left-to-right or in right-to-left order of the variables.
Now, for a function $f(x_{n-1},\ x_{n-2},\ldots,x_0)$ of n variables, the number of possible strings of elements is 2^n. From the above definition of *RSBF*, the value of $f(x_{n-1},\ x_{n-2},\ldots,x_0)$ will be the same for a set of input strings of elements,where any input string is obtained by cyclic permutation of some other input string. A set of strings of elements 0 and 1 of n variables which are rotation symmetric is

said to form an *orbit*. Figure 1 illustrates the set of 14 orbits for a function of 6 variables.

{(000000)}	orbit 0
{(000001) (000010) (000100) (001000) (010000) (100000)}	orbit 1
{(000011) (000110) (001100) (011000) (110000) (100001)}	orbit 2
{(000101) (001010) (010100) (101000) (010001) (100010)}	orbit 3
{(000111) (001110) (011100) (111000) (110001) (100011)}	orbit 4
{(001001) (010010) (100100)}	orbit 5
{(001011) (010110) (101100) (011001) (110010) (100101)}	orbit 6
{(001101) (011010) (110100) (101001) (010011) (100110)}	orbit 7
{(001111) (011110) (111100) (111001) (110011) (100111)}	orbit 8
{(010101) (101010)}	orbit 9
{(010111) (101110) (011101) (111010) (110101) (101011)}	orbit 10
{(011011) (110110) (101101)}	orbit 11
{(011111) (111110) (111101) (111011) (110111) (101111)}	orbit 12
{(111111)}	orbit 13

Fig. 1. Orbits of $RSBF$ for $n = 6$

Each *orbit* is also known as a *partition*. Number of partitions g_n satisfies the inequality

$$g_n \geq 2 + \frac{2^n - 2}{n} \tag{1}$$

where the equality holds when n is prime[3]. We now introduce a few definitions for the subsequent discussions.

Definition 1. *Hamming distance $d(s_1, s_2)$ between two binary strings s_1 and s_2 is the number of positions where the bit values of s_1 and s_2 differ.*

Definition 2. *If a boolean function $f(x_{n-1}, x_{n-2},\ldots,x_0)$ exhibits rotation symmetry, then the period over which it exhibits this property is defined to be the* cycle length *for the function.*

Consider a rotation symmetric boolean function whose associated binary strings are {(1011), (0111), (1110), (1101)} obtained through all possible cyclic permutation of the bits containing 3 ones and 1 zero.The $RSBF$ $f(x_3, x_2, x_1, x_0)$ must have the same value for all the permutations. Cycle length of this function is then 4.

Definition 3. *For a n-variable* RSBF *containing g_n orbits,the orbits which have maximum cycle length i.e. cycle length n are known as long cycles. If the cycle length is some factor of n then it is known as a short cycle.*

For instance, in Figure 1, the *RSBF*s corresponding to orbits 1, 2, 3, 4, 6, 7, 8, 10 and 12 are all long cycles, whereas those corresponding to orbits 0, 5, 9, 11, and 13 are all short cycles. The following observation is then clear:

Observation 1 *For a* RSBF *of n variables, n > 1, the two trivial orbits containing all zeros and all ones respectively are always short cycles.*

Lemma 1 *If n is prime, then* $\binom{n}{r}$ *can be expressed as the product of the number n and an integer.*

Proof. Let us assume the result is wrong.So if n is prime, then $\binom{n}{r}$ can be expressed as the product of the number n and a fraction. $\binom{n}{r}=\frac{n!}{(n-r)!\times r!}=n\times\frac{(n-1)!}{(n-r)!\times r!}$. It is quite trivial that $\binom{n}{r}$ yields an integer. So to get the value of $\binom{n}{r}$ as integer either the denominator of $\frac{(n-1)!}{(n-r)!\times r!}$ is n or the value of $\frac{(n-1)!}{(n-r)!\times r!}$ is an integer.Since n is a prime number so it can not be expressed as product of two or more numbers other than 1 and the number n itself.So the denominator of $\frac{(n-1)!}{(n-r)!\times r!}$ i.e. $(n-r)!\times r!$ will never be n.Hence it is a contradiction. So $\frac{(n-1)!}{(n-r)!\times r!}$ is not a fraction. ■

Lemma 2 *For a* RSBF *of n variables, if n is prime, then there does not exist any orbit of cycle length less than n except for the two trivial orbits containing all zeros and all ones.*

Proof. Let r denotes the number of 1s in the binary string of n bits. Then, total number of binary strings that can be formed with n bits having r ones is $\binom{n}{r}$. Since n is prime, $\binom{n}{r}$ can be expressed as the product of the number n and an integer (by Lemma 1). Moreover, the cycle length of the *RSBF* cannot be greater than n. Thus, every cycle that can be generated for the *RSBF* are of full cycle length except for the two trivial orbits containing all zeroes and all ones. ■

Lemma 1 and Lemma 2 clearly indicates that for a composite value of n, the different strings generated for the *RSBF* will have some short cycles. A closer look at the Boolean strings corresponding to *RSBF* yields the following observation:

Observation 2 *If the cycle length of an orbit for a n-variable* RSBF *is less than n, then there always exists a substring within the corresponding binary strings which is also repeated within every string of the orbit.*

The length of this repeating substring is defined to be the *internal period* of the orbit. In Figure 1, for example, for orbit 5, cycle length is 3, and the substring {001} is repeated three times in every element of the orbit, and the internal period is also 3. A formal definition of *internal period* is given below:

Definition 4. *For a n-variable binary string corresponding to a* RSBF,*where n is not prime if d be a divisor of n, then a substring of d bits of the n- variable string is said to exhibit periodicity d if* $(a_{n-1}, \ldots, a_{n-d}) = (a_{n-d-1}, \ldots, a_{n-2d}) = \ldots = (a_{d-1}, \ldots, a_0)$. *d is defined to be the internal period of these substrings.*

For example in Figure 1 orbit 5 is {(001001)(010010)(100100)},where substring (001) has periodicity 3. The following observation follows from the Definition 4.

Observation 3 *Product of the internal period and the number of substrings within a string yields the number of variables of the string.*

For instance, in Figure 1, for *Orbit 9*, number of substrings is 3,*internal period* is 2 and the *number of variables* is 6.

3.1 Algebraic Normal forms and *RSBF*

The classical approach to the analysis, synthesis or testing of a switching circuit is based on the description by the Boolean algebra operators. A description of a switching circuit based on Modulo-2 arithmetic (the simplest case of the Galois field algebra [10]) is inherently redundancy-free, and is implemented as the multi-level tree of XOR (addition operator over $GF(2)$) gates.

Definition 5. *An n-variable Boolean function $f(x_{n-1}, \ldots, x_1, x_0)$ can be expressed as a multivariate polynomial over $GF(2)$. More precisely, $f(x_{n-1}, \ldots, x_1, x_0)$ can be written as : $f(x_{n-1}, \ldots, x_1, x_0) = a_0 \bigoplus \Sigma_{i=1}^{n} a_i x_i \bigoplus \Sigma_{1 \le i < j \le n} a_{ij} x_i x_j \ldots \bigoplus a_{1,2,\ldots,n}$, where the coefficients $a_0, a_i, a_{ij}, \ldots, a_{1,2,\ldots,n} \in \{0, 1\}$ and $x_i \in \{0, 1\}$. This representation of $f(x_{n-1}, \ldots, x_1, x_0)$ is known as its* Algebraic normal form (ANF) *or its* Positive Polarity Reed-Muller form (PPRM).

The number of variables in the highest order product term with non-zero coefficient of an ANF is its *algebraic degree*. A Boolean function is defined to be *homogeneous* if all terms of its ANF are of the same degree.

Definition 6. *A homogeneous function of degree one is called a* linear *function.*

Definition 7. *A Boolean function of degree at most one is called an* affine *function.*

Definition 8. *The non-linearity NL_f of a Boolean function $f(x_{n-1}, x_{n-2}, .., x_0)$ is the minimum number of truth table entries that must be changed in order to convert $f(x_{n-1}, x_{n-2}, .., x_0)$ to an affine function.*

Non-linearity of a Boolean function $f(x_{n-1}, x_{n-2}, .., x_0)$ may be measured as the minimum Hamming distance between truth tables of $f(x_{n-1}, x_{n-2}, .., x_0)$ and its corresponding (same degree) affine function [9].

For a Boolean function $f(x_{n-1}, x_{n-2}, .., x_0)$ of n variables, there are 2^n different input values corresponding to the function. From the definition of *RSBF*, the function has the same value corresponding to each of the subsets generated from the rotational symmetry. Each of these subsets is called a *partition*. In the following Section, we propose three algorithms for the generation of partitions for rotational symmetry for a given number n variables. For each partition (orbit), we select the first element of the partition as its representative element.

4 Proposed Algorithms

The proposed algorithm starts with a string of n zeros, which forms the first orbit. Subsequent representative strings are formed by the odd numbers whose binary representation has r initial zeros starting with most significant bit, followed by $(r+1)^{th}$ bit as 1 and least significant bit as 1. Left rotation of these starting (representative) strings by up to r bits yields an even number. Further left rotation may yield again an odd number.

Lemma 3 *If a n bit odd number (d) consists of* 1^{st} *r bits as zeros ,starting with most significant bit,the* $(r+1)^{th}$ *bit as one, and the least significant bit as one, then* $d \times 2^{r+1} > 2^n$*, where* $n > r \geq 1$*.*

Proof. From mathematical inequality, we have $2^{n-r-1} < (2^{n-r}-1)$. Now, $(2^{n-r}-1)$ is the maximum odd number (d_{max}) of $(n-r)$ bits. Prefixing r zeros to the left of binary form of d_{max} also yields d_{max}. Thus, $2^{n-r-1} < d_{max}$, i.e., $d_{max} \times 2^{r+1} > 2^n$. Moreover, the minimum odd number of $(n-r)$ bits is $(2^{n-r-1}+1)$. Let d_{min} denotes this number. It can be easily shown that $d_{min} \times 2^{r+1} > 2^n$. Thus, all odd numbers between d_{min} and d_{max} satisfy the same inequality constraint. ■

Consider an odd number whose binary representation has r initial zeros starting with most significant bit, followed by $(r+1)^{th}$ bit as 1 and least significant bit as 1. Left rotation of these starting (representative) strings by up to r bits yields an even number. Thus, from Lemma 3, we find that for the representative string s of an orbit, beyond left rotation of a number by r bits starting with s, the number obtained is not necessarily an odd number. Value of this bit string is obtained from Lemma 3. An AVL tree is used to store the Decimal values of all odd numbers from this position up to $n-1$ rotations for avoiding possible repetition in future orbit generations.

A formal description of the proposed Algorithm A is given in Figure 2.

The following result clearly follows from the description of the proposed *Algorithm A*.

Lemma 4 *Worst-case time complexity of Algorithm A is* 2^n*.*

Lemma 5 *The proposed Algorithm A requires a maximum space of* $2^{n-1} - g_n$*.*

Proof. For an n-variable string, the corresponding decimal numbers range between 0 and $2^n - 1$. Of these, the number of odd decimal values is $\frac{2^n}{2}$. We store only odd numbers in the AVL tree. Thus, the total storage requirement is $2^{n-1} - g_n$. Total number of orbits for an n-variable $RSBF$ is $g_n \geq 2 + \frac{2^n-2}{n}$. Since nodes are gradually deleted from the AVL tree, we have space requirement $\leq 2^{n-1} - g_n$. ■

The sequence of outputs generated by the proposed *Algorithm A* is illustrated with an example below.

Consider $n = 5$. Thus, the maximum value of the corresponding decimal number is 31. *Orbit 1*: All zeros.

Algorithm A

Data structures: $Cntr$:# of orbits, $Result[]$: starting string of orbit, T: AVL tree
Input: Number of bits
Output: Starting string of every orbit and total number of orbits

1. Initialize $Cntr = 0$;
2. Take the string of all zeros as Orbit 0;
   store its decimal form in $Result[]$;
3. $Cntr = Cntr + 1$;
4. Take the string of first $n-1$ *zero* bits and last bit *One* as Orbit 1;
   store its decimal form in $Result[]$;
5. while $last < 2^n$ do
6. Take the binary string s corresponding to next odd number
7. if $s \notin T$ then
8. store decimal form d of s in $Result[]$;
9. $Cntr = Cntr + 1$;
10. while s does not reappear do
11. bit-wise rotate the binary representation of s
    to generate a set of strings $S = \{s_k | k = 1, n-1\}$
12. if $d \times 2^k > 2^n$ then store decimal form of s_k in T
    if s_k is odd, where $k = r + 1$ (by Lemma 3)
13. endwhile
14. endwhile
15. end

Fig. 2. Algorithm A for generating orbits for n-variable $RSBF$

Orbit 2: Starting string must have 4 zeros, and a one, which is decimal 1 $\{i.e.0^41\}$. Now, check for some k for which $1 \times 2^k > 31$, $k = 1, \ldots 4$. No such value of k exists.

Orbit 3: Starting bit string of this orbit must be the next odd number. Since any rotation of 1 in the starting string of *Orbit 2* yields an even number, the starting string of this orbit must have three consecutive zeros, followed by two ones (decimal 3). The condition $3 \times 2^k > 31$, $k = 1, \ldots, 4$ is satisfied for $k = 4$. Rotate the starting string four times to obtain 10001, having decimal value 17. Since decimal value of this string is odd, it is saved in the AVL tree (Figure 3(a)).

Orbit 4: Starting bit string of this orbit must be the next odd number (after starting string of previous orbit), having decimal value 5, i.e., two zeros followed by 101. For $k = 3$, $5 \times 2^k > 31$. Rotate 00011 three and four times respectively to yield 01001 (=9) and 10010 (=18). Since the number 9 is odd, it is saved in the AVL tree (Figure 3(b)).

Orbit 5: Starting bit string of this orbit must be the next odd number 7 (after 5), i.e., two zeros followed by 111. For $k = 3$, $7 \times 2^k > 31$. Rotate 00111 three and four times respectively to yield 11001 (=25) and 10011 (=19). Since both these numbers are odd, they are saved in the AVL tree (see (Figure 3 (c)).

Orbit 6: Starting bit string of this orbit must be the next odd number after 7 which is not already in the AVL tree. Since the next odd number 9 is already in the AVL tree, the starting string of this Orbit would be 01011 (=11). The

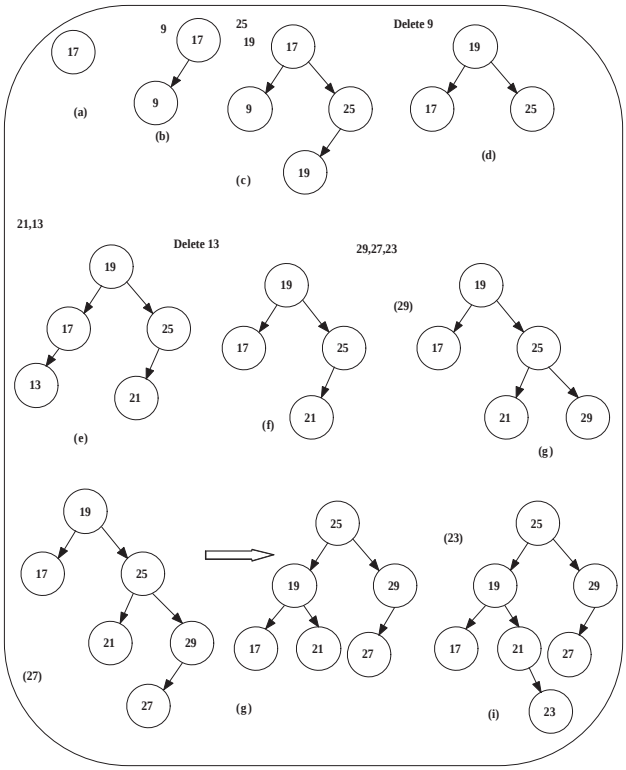

Fig. 3. An example execution of Algorithm A

number 9 is deleted from the AVL tree (Figure 3(d)) For $k = 2$, $11 \times 2^k > 31$. Rotate 01101 (=13) two to four times to yield respectively 01101 (=13), 11010 (=26), and 10101 (=21). The odd values 13 and 21 are stored in the AVL tree (Figure 3(e)).

Orbit 7: Starting bit string of this orbit must be the next odd number after 11 which is not already in the AVL tree. Since the next odd number 13 is already in the AVL tree, the starting string of this Orbit would be 01111 (=15). The number 13 is deleted from the AVL tree (Figure 3(f)) For $k = 2$, $15 \times 2^k > 31$. Rotate 01111 (=15) two to four times to yield respectively 11101 (=29), 11011 (=27), and 10111 (=23). The odd values 29, 27 and 23 are stored in the AVL tree (Figure 3(g)(h)(i)).

Orbit 8: Starting bit string of this orbit must be the next odd number after 15 which is not already in the AVL tree. Since the next few odd numbers 17, 19, 21, 23, 25, 27, and 29 are already in the AVL tree, these numbers are successively deleted from the AVL tree (see Figure 3g). Thus, the string for this Orbit 8 is 31. The algorithm terminates successfully at this point.

Figure 3 illustrates some of the steps using the AVL tree during execution of Algorithm A with an example. Let the symbols RR and LR respectively denote right and left rotations of a bit string by one bit position. For a string of bits s, we define its *rotation cousin* to be the bit string obtained on application of LR or RR on s. The following result is used to design an improved algorithm.

Lemma 6 *Right rotation of each of the bit strings corresponding to two consecutive odd integers by one bit yields two consecutive integers.*

Proof. Let two successive n-bit binary odd numbers be represented as $\{a_{n-1}, a_{n-2}, \ldots, a_1, a_0\}_2$ and $\{b_{n-1}, b_{n-2}, \ldots, b_1, b_0\}_2$ and $\{a_{n-1}, a_{n-2}, \ldots, a_1, a_0\} + 2 = \{b_{n-1}, b_{n-2}, \ldots, b_1, b_0\}$.
Now RR $\{b_{n-1}, b_{n-2}, \ldots, b_1, b_0\}$ by 1 bit yields $\{b_0, b_{n-1}, b_{n-2}, \ldots, b_1\}$.
$= b_0 2^{n-1} + (b_{n-1}2^{n-2} + b_{n-2}2^{n-3} + ... + b_1 2^0)$
$= b_0 2^{n-1} + \frac{(b_{n-1}2^{n-1} + b_{n-2}2^{n-2} ... + b_1 2^1)}{2}$
$= b_0 2^{n-1} + \frac{(b_{n-1}2^{n-1} + b_{n-2}2^{n-2} ... + b_1 2^1) + b_0 - b_0}{2}$
$= b_0 2^{n-1} + \frac{(b_{n-1}2^{n-1} + b_{n-2}2^{n-2} ... + b_1 2^1 + b_0) - b_0}{2}$
$= b_0 2^{n-1} + \frac{(a_{n-1}2^{n-1} + a_{n-2}2^{n-2} ... + a_1 2^1 + a_0) + 2 - b_0}{2}$
$= a_0 2^{n-1} + \frac{(a_{n-1}2^{n-1} + a_{n-2}2^{n-2} ... + a_1 2^1 + a_0) + 2 - b_0}{2}$ $[a_0 = b_0, \textit{since both are odd numbers}]$
$= a_0 2^{n-1} + \frac{(a_{n-1}2^{n-2} + a_{n-2}2^{n-3} ... + a_1 2^0) \times 2 + 2 + a_0 - b_0}{2}$
$= a_0 2^{n-1} + (a_{n-1}2^{n-2} + a_{n-2}2^{n-3} ... + a_1 2^0) + 1$
$= \{a_0 a_{n-1}, a_{n-2}, \ldots, a_1\}_2 + 1$. ■

Let $\{a_{n-1}, a_{n-2}, \ldots, a_1, a_0\}$ represent a n-bit string, where a_i represents a bit, $i = 0, \ldots, n-1$. Then Lemma 6 may be expressed as $\{\{a_{n-1}, a_{n-2}, \ldots, a_1, a_0\}$ RR $1\}$ $= \{\{a_{n-1}, a_{n-2}, \ldots, a_1, a_0\} + 2 \}$ RR 1.

4.1 An improved Algorithm

The proposed *Algorithm A* is improved with a minor modification. We note the following observation:

Consider the bit string (for an odd number) having 1 at the right-most position$\{0^{n-1}1\}$. Let this bit string be right-rotated right by 1-bit (i.e., n-bits left-rotate) to form a new bit string P, say $P = \{1^1 0^{n-1}\}$. The starting bit-string of each orbit is an odd number, generated by simply adding 2 to the starting string of the previous orbit. In the previous algorithm, we had to check all these starting strings in an AVL tree to avoid repetitive occurrences of numbers.

Lemma 7 *If the starting string of an orbit is P + 1, then the numbers between P + 1 and the number generated by one-bit right-rotation of (P - 1) may be ignored for generating subsequent orbits.*

Proof. From Algorithm A it is clear that the starting string of the orbits must be odd number.From the previous lemma 6 it is clear that the last rotation cousins are the consecutive numbers if starting strings of the orbits are consecutive odd numbers.When some number misses in the orbit then its corresponding last rotation cousin do no appear.So as soon as the starting string of the orbit becomes P+1 which comes already as the last cousin of some orbit we do not cosier from this to the last cousin of P - 1. ■

The above lemma shows that the n-bit left-rotation (= 1-bit right rotation) of successive odd numbers results in successive numbers. Thus, whenever the odd number becomes (P + 1), all the successive numbers up to the number which is RR of (P - 1) already appears.

A formal description of the proposed Algorithm B is given in Figure 4.

Following result is clear from the description Algorithm B.

Lemma 8 *Worst-case time complexity of Algorithm B is 2^n.*

Lemma 9 *The proposed Algorithm B requires a maximum space of $2^{n-1} - g_n - a$, where $a = RR(2^{n-1} + 1) - (2^{n-1} + 1)$.*

Proof. Follows from Lemma 6. ■

4.2 A further improved Algorithm

In both the algorithms proposed above, an AVL tree is used to reduce certain iterations. The following observation helps in getting rid of this auxiliary data structure and its associated operations.

Observation 4 *If the rotation cousin of an odd starting number of an orbit is also odd, and is greater than the value of the next starting string of the next orbit, then this starting string may be discarded.*

A formal description of Algorithm C is given in Figure 5.

Following result is clear from the description Algorithm C.

Algorithm B

Data structures: $Cntr$:# of orbits, $Result[]$: starting string of orbit, T: AVL tree
Input: Number of bits
Output: Starting string of every orbit and total number of orbits

1. Initialize $Cntr = 0$;
2.Store the decimal value of 1 0^{n-1} to some variable P
3. Do not store the value from P to RR of P - 1 in T
4. Take the string of all zeros as Orbit 0;
5. store its decimal form in $Result[]$;
6. $Cntr = Cntr + 1$;
7. Take the string of first $n-1$ *zero* bits and last bit *One* as Orbit 1;
8. store its decimal form in $Result[]$;
9. while $last < 2^n$ do
10. Take the binary string s corresponding to next odd number
11. if $s \notin T$ then
12. store decimal form d of s in $Result[]$;
13. $Cntr = Cntr + 1$;
14. while s does not reappear do
15. bit-wise rotate the binary representation of s to generate a set of strings $S = \{s_k | k = 1, n-1\}$
16. if $d \times 2^k > 2^n$ then store decimal form of s_k in T if it is odd, where $k = r + 1$ (by Lemma 3)
17. if $d \geq P$ then d=RR of P - 1
18. endwhile
19. endwhile
20. end

Fig. 4. Improved algorithm B for generating orbits for n-variable $RSBF$

Lemma 10 *Worst-case time complexity of Algorithm C is* 2^n.

The space requirement for Algorithm C is much less than those of the previous algorithms, as it does not use the AVL tree.

The proposed Algorithm C is illustrated with an example below.

Consider $n = 7$. Thus, the maximum value of the corresponding decimal number is 31. *Orbit 1*: All zeros.

Orbit 2: Starting string will be $\{0^6 1\}$ as none of the rotational cousins of $\{0^6 1\}$ is of smaller value.

Orbit 3: Starting bit string of this orbit must be the next odd number, i.e., the bit-string $\{0^5 1^2\}$, as none of the rotational cousins of $\{0^6 1\}$ is smaller than $\{0^5 1^2\}$.

Orbit 4: Starting bit string of this orbit must be the next odd number (= 5), i.e., $\{0^4 101\}$ as none of the rotational cousins of $\{0^5 1^2\}$ is smaller than $\{0^4 101\}$.

Continuing in this manner, we find that up to *Orbit 9* the consecutive odd numbers form the starting strings of the respective orbits. Thus, the starting string of *Orbit 9* is $\{0^3 1^4\}$. The next odd number is 17, i.e., $\{0^2 10^3 1\}$. We observe that 3^{rd} left-rotate cousin of $\{0^2 10^3 1\}$ is $\{0^3 10^2 1\}$ (=9), which is less than 17. Thus, we discard the number 17, and do not consider it as the starting number of any orbit. For the next consecutive odd numbers between 19 nd 23, none of them

Algorithm C

Data structures: $Cntr$:# of orbits, $Result[]$: starting string of orbit
Input: Number of bits
Output: Starting string of every orbit and total number of orbits

1. Initialize $Cntr = 0$;
2. Take the string of all zeros as Orbit 0;
   store its decimal form in $Result[]$;
3. $Cntr = Cntr + 1$;
4. Consider the bit-string s corresponding to the next odd number;
   store its decimal form in $Result[]$;
5. while bit-string corresponding to $s \neq \{1^n\}$ do
6. if a number p has any of its rotational cousins $p_c < p$ then goto Step 8
7. take bit-string corresponding to s as the starting string of the next orbit
8. consider the bit string corresponding to the next odd number 9. endwhile
10. end

Fig. 5. Improved algorithm (without AVL tree) for generating orbits for n-variable $RSBF$

have any smaller left-rotate cousin, and hence, they form the starting numbers for the next three successive orbits. The algorithm continues in this manner, and terminates successfully at *Orbit 20*.

5 Implementation of the Proposed Algorithms

The proposed algorithms have the same worst-case time complexities, and exhibit drastic improvement in space requirements from Algorithm A through Algorithm C. Algorithms A and B use the AVL tree, while the other two do not require this data structure. Algorithm A was implemented in C language on a 32-bit PC running on Intel CPU under Windows environment having 2.27 GHz clock speed. It could generate $RSBF$s up to 26 bits in reasonable time.

6 Conclusion

In this work, we investigated rotation symmetric Boolean functions. Our major contributions include designing of certain efficient algorithms for generation of $RSBF$s. Future possibilities of work include (i) possible reducing of the time complexities, (ii) generating $RSBF$s with more than 30 variables, and (iii) generating cryptographically better $RSBF$s. The proposed algorithms are of great importance as they can help to construct new cryptographically very strong boolean functions.

References

1. Zvi Kohavi, *Switching and Finite Automata Theory*, 2^{nd} Edition, Tata-McGraw Hill, 2008.

2. J. Pieprzyk and C. X. Qu, *Fast hashing and Rotatio-symmetric functions*, Journal of Universal Computer Science, pp. 20-31, vol. 5, no. 1, 1999.
3. P. Stanica and S. Maitra, *Rotation Symmetric Boolean functions - Count and Cryptographic properties*, Discrete Applied Mathematics, vol. 156, no. 10, May, 2008.
4. P. Stanica and S. Maitra and J A Clark, Results on Rotation symmetric Bent and Correlation immune Boolean functions in B. Roy and W. Meier (Eds.), FSE 2004, LNCS 3017, pp. 161-177, 2004, International Assoc. for Cryptologic Research.
5. M.Hell, A. Maximov,and S. Maitra, *On efficient implementation of search strategy for RSBF*, 9^{th} International workshop on Algebraic and Combinatorila coding theory, ACCT 2004, June 19-25, 2004, Bulgaria.
6. P.Sarkar and S. Maitra, *Construction of rotation symmetric boolean function with important cryptographic properties*,Advances in cryptology,EUROCRYPT-2000,Lecture notes in Computer Science,Vol-1807,Berlin 2000,PP 485-506.
7. ShaoJing Fu, Chao Li and LongJiang Qu, *On the number of rotation symmetric Boolean functions*, Science China Information Sciences March 2010, vol. 53, no 3, pp. 537?545.
8. A Canteaut and M Videau, *Symmetric Boolean functions*, IEEE Transactions on Information Theory, vol. 51, no. 8, pp. 2791-2811, Aug 2005.
9. J Butler and T Sasao, *Logic functions for cryptography - A tutorial*, Proceedings of Reed-Muller Workshop, Okinawa, Japan, 2010, pp. 127-136.
10. N. Deo, *Graph Theory with Applications to Engineering and Computer Science*, PHI Learning Pvt. Ltd., 2004.
11. A menezes, p Van Oorschot,S Vanstone, *Handbook of applied cryptography*,1997,CRC Press,Inc.
12. X Zhang, Hua Guo,Yifa Li,Proof of a Cojecture about Rotation Symmetric Functions.