Evolutionary Design of Trace Form Bent Functions

Min Yang, Qingshu Meng, and Huanguo Zhang

school of computer science, Wuhan university, Wuhan Hubei, China
yangm75@hotmail.com, mqseagle@yahoo.com, liss@whu.edu.cn

Abstract. In order to design bent functions, evolutionary algorithm based on truth table, algebraic normal form or Walsh spectra are already known. Evolutionary algorithm based on trace function form is not known to authors' knowledge. In this paper, we give an evolutionary algorithm based on the trace representation of boolean function. With the algorithm, we constructed many bent functions and made some analysis work. First we observe that all 3 affinely inequivalent bent classes in 6-variable can be written as the linear sum of 2 or 3 monomial trace functions. We make a conclusion that affine transform can be used to change the linear span, which lead to a method constructing perfect nonlinear s-box of non-Niho type; Second, we find that some exponents take more chances to construct bent functions while some exponents take less. By this observation, we give each exponent a cost function, which make our algorithm more efficient than exhaustive searching algorithm or random algorithm. This is also the advantage over the algorithms based on the algebraic normal form, truth table, or Walsh spectra because we don't know what kinds of algebraic normal form, truth table, Walsh spectra are more possible to be used to construct bent functions; Third, in order to make a comparison with Fuller's algorithm, we classify them into affinely inequivalent classes. Present experiments show we obtained much more classes than Fuller's algorithm.
keyword evolutionary algorithm, trace function, bent functions.

1 Introduction

Since the concept of bent function was proposed by Rothaus in 1976[1], there are many papers, like [1–6], discussing bent function for interest in itself and for its wide applications. In order to be resistant against the differential cryptanalysis [7], a boolean function should have low absolute autocorrelation. In order to be resistant against linear cryptanlysis[8], a boolean function should have high nonlinearity. Bent functions have highest nonlinearity and zero autocorrelation. Therefore it can be used in cryptography area, especially in symmetric cipher to construct S-box. It can also be used spread communication [9], where Low peak-to-average power ratio is critical to multi-code code-division multiple-access(MC-CDMA). Bent functions are used as code words to obtain the lowest peak-to-average power ratio. Especially, in order to construct good codes, we

need construct bent functions as many as possible. To enumerate, construct and classify bent function is a longstanding researching topic and old open topic[9, 10].

Recently evolutionary computing is introduced to design bent functions[11–13] and is proved to be successful. In [11], using the algebraic normal form of boolean function as evolutionary object, or more exactly using the quadratic bent functions as the initial evolutionary object, Fuller designed many affinely inequivalent bent functions in 8-16 variables. In paper[12], using the Walsh spectra as evolutionary object, Clark proposed an algorithm designing 8-variable bent functions efficiently. In paper [13], using the truth table as evolutionary object, Meng gave an algorithm designing almost all 6-variable bent functions. However the latter two methods are both inefficient for 10 or more variables. Among the four kinds of representation of boolean function: truth table, algebraic normal form, Walsh spectra and trace form representation, the first three representations have been used as the evolutionary objects. To the authors' knowledge, algorithm using the trace function as evolutionary object is not known before.

In this paper, boolean functions are represented as the linear sum of trace functions. Using the trace form representation as evolutionary object, we give an efficient algorithm to design bent functions. With the constructed bent functions, we made a lots of analysis work. First we observe that all 3 affinely inequivalent bent functions can be written as the linear combination of 2 or 3 monomial trace functions. We make a conclusion that affine transform can be used to change the linear span. By this observation, we give a method to construct perfect nonlinear s-box of non-Niho type; Second, by observing the trace form of constructed bent functions, we find that some exponents take more chances to construct bent functions while some exponents take less. By this observation, we give each exponent a cost function, which make our algorithm more efficient than purely exhaustive searching algorithm. This is also the advantage over the algorithms based on the algebraic normal form, truth table, or Walsh spectra because we don't know what kinds of algebraic normal form, truth table, Walsh spectra are more possible to be used to construct bent functions; Third, our algorithm is so efficient that we can obtain hundreds of bent functions in just one second. In order to make a comparison with Fuller's algorithm, we also classify them into affinely inequivalent classes. Present experiments show we obtained much more affinely inequivalent bent functions than Fuller's algorithm.

The rest of the paper is organized as follows. In Section two, we recite some necessary definitions and notations. In Section three, we give our evolutionary algorithm and the evolutionary results. In Section four, we have an analysis on the obtained bent functions and get some interesting results. Finally a short conclusion is made in Section 5.

2 Preliminary

Denote by F_2 the field with two elements and by F_{2^n} the field with 2^n elements. Let $F_{2^n}^* = F_{2^n} \setminus 0$.

Let $Q = 2^n - 1$. We call an integer r is a coset leader modulo Q if r is the minimum integer in set

$$\{r2^i \ mod \ Q | i = 0, 1, \cdots, n-1\},$$

and its size is denoted by $L(r)$. Denote by $CS = \{r | r \text{is a coset leader}\}$, the set of coset leaders modulo Q.

The map $tr_1^n(x)$ from F_{2^n} to F_2 is called trace function, defined as

$$tr_1^n(x) = x + x^2 + x^4 + \cdots + x^{2^{n-1}}.$$

If n is known in context, the trace function can be simply denoted by $tr(x)$. Some properties are listed below, refer to [14] for more.

1. $tr(\alpha + \beta) = tr(\alpha) + tr(\beta)$, for all $\alpha, \beta \in F_{2^n}$.
2. $tr(c\alpha) = c \cdot tr(\alpha)$, for all $c \in F_2, \alpha \in F_{2^n}$.
3. $tr(\alpha^2) = tr(\alpha)$.

If boolean function is define as $F_{2^n} \to F_2$, it can be represented in trace form:

$$f(x) = \sum_{d \in CS} tr(\alpha x^d), \alpha \in F_{2^n}^*, x \in F_{2^n} \tag{1}$$

In formula (1), each trace function is called a monomial, d is called exponent and $\sum_{d \in CS} L(d)$ is called the linear span of $f(x)$.

If boolean function is define as $F_2^n \to F_2$, it can be represented in algebraic normal form:

$$f(x) = \sum_{s \in F_2^n} a_s x^s, a_s \in F_2, x \in F_2^n, x^s = x_0^{s_0} x_1^{s_1} \cdots x_{n-1}^{s_{n-1}}. \tag{2}$$

In formula (2), x^s is also called monomial, and the degree of $f(x)$ is defined as

$$deg(f) = max_{a_s \neq 0} H(s),$$

where $H(s)$ is the Hamming weight of s.

There is a one-to-one correspondence between the above two representations. Let α be a primitive element of the field F_{2^n} and let $(1, \alpha, \cdots, \alpha^{n-1})$ be a basis of F_{2^n} over F_2, then the elements of F_{2^n} can be represented as

$$x_0 + x_1\alpha + x_2\alpha^2 + \cdots + x_{n-1}\alpha^{n-1},$$

where $x_i \in F_2$, for $i = 0, 1, \cdots, n-1$. Now we have a map $F_{2^n} \leftrightarrow F_2^n$ defined by

$$x_0 + x_1\alpha + x_2\alpha^2 + \cdots + x_{n-1}\alpha^{n-1} \leftrightarrow (x_0, x_1, \cdots, x_{n-1}).$$

This correspondence induce the correspondence between the trace form and algebraic normal form. That is,

$$f(x) = f(\sum_{i=0}^{n-1} x_i \alpha^i) \leftrightarrow g(x_0, x_1, \cdots, x_{n-1}). \tag{3}$$

Refer to [15] for detail.

For description convenience, we usually use an integer to represent a vector $x \in F_2^n$. For example, in F_2^3,

$$\begin{array}{c} 000 \leftrightarrow 0 \\ 001 \leftrightarrow 1 \\ \cdots\cdots \\ 111 \leftrightarrow 7 \end{array}.$$

Let x take value $0, 1, \cdots, 2^n - 1$, we get a sequence $(f(0), f(1), \cdots, f(2^n - 1))$, and it is called the truth table of the function $f(x)$.

Definition 1 *Let $f(x), x \in F_2^n$ be a boolean function. Define*

$$s_f(w) = \sum_{x \in F_2^n} (-1)^{f(x)} (-1)^{w \cdot x}$$

as the Walsh spectrum of $f(x)$ at vector w, where $w \in F_2^n$.

Definition 2 *[1] Let $f(x), x \in F_2^n$ be a Boolean function. If for any $\omega \in F_2^n$, $s_f(\omega) = \pm 2^{n/2}$, then $f(x)$ is called bent function.*

Definition 3 *[17]If an integer $d < 2^n - 1$ such that*

$$d = 2^i \mod 2^k - 1,$$

then d is called Niho exponent, where $i < n$, and $n = 2k$.

If all the exponents in a trace form bent function are of Niho type, then the bent function is called a Niho-type bent function.

Definition 4 *[16] Vectorial function $F(x) = \{f_1(x), f_2(x), \cdots, f_k(x)\}$ is called vectorial bent function or perfect nonlinear s-box if any nonzero linear sum of $f_i(x), i = 1, 2, \cdots, k$ is a bent function, where $k \leq n/2$, $x \in F_2^n$, and each $f_i(x)$ is a boolean function.*

For its application in cryptography, perfect nonlinear s-box is discussed in papers[16, 18].

The derivative function of $f(x)$ in direction v is defined as $f_v(x) = f(x) + f(x + v)$.

If S is a set, then the notation $|S|$ denote the size of the set S.

In this paper, we always let $n = 2k$, an even positive integer.

3 The Algorithm

Evolutionary computing is inspired by the evolution of nature and used to solve problem which is not known or not known thoroughly. There are two key steps. One is the evolutionary strategy , like how to code, how to mutate and mate,

which offspring to be reserved. The other key step is the cost function, which judge if a gene, a mutation or mating is good or not.

In the paper, we represent boolean function in trace form. That is,

$$f(x) = \sum_{i=0}^{|CS|} tr(\alpha_i x^{d_i}),$$

where $d_i \in CS, \alpha_i \in F_{2^n}^*$. As the degree is at most k for a n variables bent function, so only the coset leader d_i in CS satisfying the condition

$$H(d_i \bmod 2^{L(d_i)} - 1) \leq k$$

can be used to construct bent functions. Denote by BCS the set of coset leaders satisfying the above condition.

From trace representation, if we only consider boolean functions with not too much monomials, we can reduce the computation complexity. What'more, by giving each exponent a cost function $C(d_i)$, our algorithm is better than exhaustive searching algorithm in term of efficiency. This can be seen from the following Section 4. The algorithm is as follows (take $tr(\alpha^i x^{d_1} + \alpha^j x^{d_2})$ as example):

Algorithm 1 *Input: n, the number of variable; Output:bent functions.*

$NN1 = 2^n$;
int r[NN1],s[NN1],t[NN1];
for($d_1 \in BCS$) $C(d_1) = 0$;
for($d_1 \in BCS$)
{
  for($d_2 \in BCS$ and $d_2 > d_1$)
  if($C(d_1) > \lambda_1$ and $C(d_2) > \lambda_2$)
  {
    for($i = 0; i < NN1 - 2; i++$)
    {
      tratofun(i, d_1, r);
      for($j = 0; j < NN1 - 2; j++$)
      {
        tratofun(j, d_2, s);
        for($k = 0; k < NN1; k++$) t[k]=r[k]+s[k];
        If t is bent function, the $C(d_1), C(d_2)$ increased;
        else $C(d_1), C(d_2)$ decreased.
        If d_1 can't construct bent function for sequential d_2 above N_1 times , begin the next d_1.
        If d_2 can't construct bent function for sequential j above N_2 times, begin the next d_2.
      }
    }
  }
}

Notes: $C(d_i)$ is the cost function of the exponent d_i. If the exponent can be used to construct bent function, the value of the cost function get higher, else the value get lower. tratofun(i, d, s) calculates the truth table s of function $tr(\alpha^i x^d)$, where α is a primitive element of F_{2^n}. $\lambda_1, \lambda_2, N_1, N_2$ are four properly selected thresholds.

By the above algorithm, we can design bent functions in 6-16 variables with a few number of monomials, like $f(x) = tr(\alpha^i x^{d_1}) + tr(\alpha^j x^{d_2})$, or $f(x) = tr(\alpha^i x^{d_1}) + tr(\alpha^j x^{d_2}) + tr(\alpha^k x^{d_3})$, where α is a primitive of corresponding field. Here we give some examples.

In 6 variables case, let $F_{64} = \{0, 1, \alpha, \alpha^2, \cdots, \alpha^{62}\}$, where α is a root of $x^6+x+1 = 0$. We list some parameters (d_1, d_2) in $tr_1^6(\alpha^i x^{d_1}+\alpha^j x^{d_2})$ form bent functions: $(3, 5), (3, 9), (3, 21), (3, 27), (5, 9), (7, 9), (7, 21), (7, 27), (9, 11), (9, 21), (9, 27)$. By selecting the value of (i, j), we can get many bent functions. However they can be classified into only two affinely inequivalent bent functions $tr_1^6(\alpha^0 x^3+\alpha^5 x^5)$ and $tr_1^6(\alpha^3 x^7+\alpha^0 x^9)$. Some parameters of form (d_1, d_2, d_3) are $(3, 5, 7), (3, 5, 11), (3, 5, 13), (3, 7, 13), (5, 9, 21), (7, 13, 21), (11, 13, 21)$. From them, we get another affine inequivalent function $tr_1^6(\alpha^1 x^3+\alpha^6 x^7+\alpha^{60} x^{13})$. So we get all 3 affine inequivalent bent functions.

In 8 variable case, let $F_{256} = \{0, 1, \alpha, \alpha^2, \cdots, \alpha^{254}\}$, where α is a root of $x^8 + x^4+x^3+x^2+1 = 0$. Some parameters of form (d_1, d_2) are: $(3, 51), (5, 15), (5, 45), (9, 85), (15, 17), (15, 45), (17, 23), (25, 45), (45, 85)$. Some parameters of form (d_1, d_2, d_3) are $(3, 5, 9), (3, 9, 17), (3, 15, 27), (5, 9, 17), (5, 15, 25), (5, 25, 45), (9, 15, 21), (15, 25, 45), (17, 21, 27), (17, 25, 45)$. By change the value of (i, j) or (i, j, k), we get at least 53 affinely inequivalent bent functions, see Appendix 1. We also get some bent functions of four monomials. For example some parameters of form (d_1, d_2, d_3, d_4) are $(5, 15, 25, 45), (5, 15, 25, 85), (5, 15, 45, 85), (5, 25, 45, 85), (15, 25, 45, 85)$. By changing the coefficients, we get at least another 67 affinely inequivalent bent functions, see Appendix 1. Similarly, we can get many bent functions in 10-16 variables. For example, in 10 variables, only by changing the coefficients (i, j) for $tr_1^{10}(\alpha^i x^{31} + \alpha^j x^{93})$, we can get several hundreds of affinely inequivalent bent functions. It is a very desirable property that by changing the coefficients, we can get affinely inequivalent bent functions.

4 Analysis of the results

4.1 On the Affine Transform on x

One obvious shortage of our obtained bent functions is that the number of monomials is very low or in other words, the linear span is very low. But generally, a simple form in trace representation doesn't mean a simple form in its algebraic normal norm. What's more, the shortage can be easily overcome by a simple method due to the following observation.

In 6-variable case, all 3 affinely inequivalent bent functions are : $tr_1^6(\alpha^0 x^3 + \alpha^5 x^5), tr_1^6(\alpha^3 x^7 + \alpha^0 x^9)$ and $tr_1^6(\alpha^1 x^3 + \alpha^6 x^7 + \alpha^{60} x^{13})$. But there exist a lot of bent functions with 7 or more monomials. All these functions must be got

from the 3 inequivalent functions by the affine transform on input x. That is, affine transform can change the linear span. Further, the following theorem give a theoretical analysis.

Theorem 1 *The probability of boolean functions with linear span* $2^n - 2$ *is almost 1.*

Proof For boolean function $f(x) = \sum_{r \in CS} tr(\delta_r x^r), \delta_r \in F_{2^n}$, the size of set CS can be estimated to be $k = 2^n/n$. Then the number of function of form $f(x) = tr(\delta_1 x^{r_1} + \delta_2 x^{r_2} + \cdots + \delta_i x^{r_i})$ is

$$C_k^i (2^n - 1)^i, i \leq k.$$

So the number of functions with linear span $2^n - 2$ is

$$C_k^k (2^n - 1)^{2^n/n} = (2^n - 1)^{2^n/n}.$$

As

$$\lim_{n \to +\infty} \frac{(2^n - 1)^{2^n/n}}{2^{2^n}} = 1,$$

this ends the proof.

We don't know the distribution of bent functions in the whole boolean functions space. But with experiments, we found the linear spans of most of bent functions are large.

Now we can solve the problem proposed in the first paragraph of this section. Suppose $f(x), x \in F_{2^n}$ is a sum of two or three monomials, and by formula 3 we can suppose $g(x), x \in F_2^n$ is its algebraic normal form . Let $GL(n, 2)$ be the general linear group. Given any a matrix $A \in GL(n, 2)$, $g(xA)$ is also a bent function, but usually there are much more monomials in its trace representation. For example, let $f(x) = tr_1^8(x^5 + x^{15} + x^{53}), x \in F_{256}$. Let $g(x)$ be its algebraic normal form. Randomly choose a matrix $A = (0xd4, 0x5d, 0x25, 0x38, 0x9, 0x7d, 0x5a, 0x3)$, where each hex number is a row vector, then the function $g(xA)$ is also a bent function, but its trace form representation is $tr_1^8(\alpha^{183}x^1 + \alpha^{105}x^3 + \alpha^{107}x^5 + \alpha^{133}x^7 + \alpha^{41}x^9 + \alpha^{234}x^{11} + \alpha^{167}x^{13} + \alpha^{100}x^{15} + \alpha^{203}x^{19} + \alpha^{141}x^{21} + \alpha^{33}x^{23} + \alpha^{25}x^{25} + \alpha^{234}x^{27} + \alpha^{199}x^{29} + \alpha^{177}x^{37} + \alpha^{68}x^{43} + \alpha^{146}x^{45} + \alpha^{123}x^{53}) + tr_1^4(\alpha^{238}x^{17} + \alpha^{238}x^{51}) + tr_1^2(\alpha^{170}x^{85})$. The fast generation of matrix A and the calculation of $f(xA)$ can be seen in Appendix 2.

By this simple method, we can construct perfect nonlinear s-box of non-Niho type. Due to Dobbertin and Leander[19] we can get a perfect nonlinear s-box $F(x) = \{f(x), f(\gamma_1 x), \cdots, f(\gamma_{k-1} x)\}$ from a Niho-type bent function $f(x)$, where $\{1, \gamma_i \in F_{2^k}, i = 1, 2, \cdots, k-1, \}$ are linearly independent. Suppose $G(x)$ is the algebraic normal form representation of $F(x)$. Obviously for any randomly given $A \in GL(n, 2)$, $G(xA)$ is still a perfect nonlinear s-box, but usually not of Niho-type. We give one example. Take $f(x) = tr_1^4(x^{17}) + tr_1^8(x^{23})$ as example and let $\gamma_0 = 1, \gamma_1 = \alpha^{17}, \gamma_2 = \alpha^{34}, \gamma_3 = \alpha^{51} \in F_{16}$ are linearly independent, where α is the same to that in Section 3. Now $F(x) = \{f(x), f(\gamma_1 x), f(\gamma_2 x), f(\gamma_3 x)\}$ is a perfect nonlinear s-box of Niho-type. Let $G(x)$ be the algebraic normal form of

$F(x)$ and randomly choose $A = (0xd4, 0x5d, 0x25, 0x38, 0x9, 0x7d, 0x5a, 0x3) \in GL(8, 2)$, where each hex number is the row vector, then $G(xA)$ is still a perfect nonlinear s-box, but is of non-Niho type. Its four output functions are listed below:

1. $tr_1^8(\alpha^{193}x^1+\alpha^{194}x^3+\alpha^{136}x^5+\alpha^{223}x^7+\alpha^{40}x^9+\alpha^{253}x^{11}+\alpha^{180}x^{13}+\alpha^{126}x^{15}+\alpha^{148}x^{19}+\alpha^{192}x^{21}+\alpha^{95}x^{23}+\alpha^{64}x^{25}+\alpha^{157}x^{27}+\alpha^{34}x^{29}+\alpha^{79}x^{37}+\alpha^{108}x^{39}+\alpha^{205}x^{43}+\alpha^{154}x^{45}+\alpha^{238}x^{53}+tr_1^4(\alpha^{17}x^{51})+tr_1^2(\alpha^0x^{85})$,
2. $tr_1^8(\alpha^{57}x^1+\alpha^{184}x^3+\alpha^0x^5+\alpha^{150}x^7+\alpha^{85}x^9+\alpha^{152}x^{11}+\alpha^{102}x^{13}+\alpha^{30}x^{15}+\alpha^{69}x^{19}+\alpha^{236}x^{21}+\alpha^{233}x^{23}+\alpha^{120}x^{25}+\alpha^{17}x^{27}+\alpha^{242}x^{29}+\alpha^{120}x^{37}+\alpha^{127}x^{39}+\alpha^{100}x^{43}+\alpha^{61}x^{45}+\alpha^{129}x^{53})+tr_1^4(\alpha^0x^{17}+\alpha^0x^{51})tr_1^2(\alpha^0x^{85})$,
3. $tr_1^8(\alpha^{250}x^1+\alpha^{99}x^3+\alpha^{90}x^5+\alpha^{12}x^7+\alpha^{17}x^9+\alpha^{32}x^{11}+\alpha^{128}x^{13}+\alpha^{166}x^{15}+\alpha^1x^{19}+\alpha^{163}x^{21}+\alpha^{130}x^{23}+\alpha^{21}x^{25}+\alpha^{221}x^{27}+\alpha^{197}x^{29}+\alpha^{17}x^{37}+\alpha^{176}x^{39}+\alpha^{102}x^{43}+\alpha^0x^{45}+\alpha^{61}x^{53})+tr_1^4(\alpha^{170}x^{17}+\alpha^{170}x^{51})$,
4. $tr_1^8(\alpha^{79}x^1+\alpha^{39}x^3+\alpha^{127}x^5+\alpha^{79}x^7+\alpha^{18}x^9+\alpha^{19}x^{11}+\alpha^{211}x^{13}+\alpha^{216}x^{15}+\alpha^{254}x^{19}+\alpha^{123}x^{21}+\alpha^{105}x^{23}+\alpha^{140}x^{25}+\alpha^3x^{27}+\alpha^{37}x^{29}+\alpha^{185}x^{37}+\alpha^{60}x^{39}+\alpha^{224}x^{43}+\alpha^{48}x^{45}+\alpha^{40}x^{53})+tr_1^4(\alpha^{221}x^{17}+\alpha^0x^{51})+tr_1^2(\alpha^0x^{85})$.

In communication area, sequence with larger linear span is expected and emphasized in many papers. As sequence can also be represented in trace form, we believe the technique we use is also useful in designing sequence with large linear span.

4.2 On the Exponents

d is called a bent exponent if there exists $\alpha \in F_{2^n}$ such that $f(x) = tr(\alpha x^d)$ is a bent function. A survey on known bent exponents is given in [19]. The known bent exponents are Gold type of form 2^r+1 (r is a natural number), Dillon type of form $2^k-1, k = n/2$[3], Dillon-Dobbertin[20] type of form $2^{2r}-2^r+1, gcd(r, n) = 1$, and the one by Canteaut of form $(2^r + 1)^2, n = 4r$.

By studying exponents in the designed bent functions, we have the following observation: If the number of monomials in bent function is relatively small, then the elements in set BCS play different roles in constructing bent functions. More exactly, when $d_1, d_2, \cdots$ is one of bent exponents, one of the multiple of bent exponent or one of the factors of bent exponent, the function $tr(\alpha^{i_1}x^{d_1} + \alpha^{i_2}x^{d_2} + \cdots)$ is more possible to be bent function, while some exponents in BCS can't be used to construct bent function.

The difference is a desirable property in evolutionary algorithm. If an element in BCS can't be used to construct bent function, only after a few number of program run, its cost function get very small. By choosing a proper threshold, it can be discarded directly in the following run of the program and thus make our algorithm more efficient than purely exhaustive searching algorithm or random algorithm. This is also the advantage over the algorithms based on the algebraic normal form, truth table, or Walsh spectra because we don't know what kinds of algebraic normal form, truth table, Walsh spectra are more possible to be used to construct bent functions;

We give two contrast examples. In 8 variables case, there are 22 coset leaders in $BCS = \{3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 37, 39, 43, 45, 51, 53, 85, 119\}$. When designing bent function of form $tr(\alpha^i x^{d_1} + \alpha^j x^{d_2})$, only after a few number of runs, the cost functions of some coset leaders, like $37, 43, 53$, get very small. By selecting proper threshold λ_1, λ_2, they are discarded directly. This is one example. The following is a contrast example. When $d_1, d_2, d_3, d_4 \in \{\text{odd multiple of Dillon exponent}\} \bigcap BCS$, for 8-16 variables, all combinations (d_1, d_2) of two monomials, all combinations (d_1, d_2, d_3) of three monomials and all combinations (d_1, d_2, d_3, d_4) of four monomials can design bent functions except the following four cases. The four cases occur in 14 variable. They are $(381, 2667), (381, 1143)$, $(1143, 2667)$ and $(381, 1143, 2667)$.

4.3 On Classification of Obtained Functions

Our algorithm is so efficient that we can obtain hundreds of bent function only in one second. In order to make a comparison with Fuller's algorithm, we classify the obtained functions under the action of affine group.

The affinely classification of boolean functions is already discussed in several papers[3, 21–23]. In order to classify the obtained bent functions, we use the method in [3, 23].

Two functions $f(x), g(x)$ are called equivalent if there exists $A \in GL(n, 2), b \in F_2^n$ such that $f(x) = g(xA + b)$. A map M from the set of boolean functions to a set of numbers is called an invariant if $M(f) = M(g)$ holds for any two equivalent functions $f(x), g(x)$. All functions with same invariant value are taken as one equivalent class. By this way, invariant is good tool to classify set. If we know T, the number of equivalent classes under some equivalent relationship, and an invariant just takes T different values, then the set is already classified. On this occasion, the invariant is called a discriminant of the set.

Proposition 2 *[3] If* $f(x) = g(xA+b), x, b \in F_2^n, A \in GL(n, 2)$, *then* $\{f_v(x)|v \in F_2^n\}$ *is affinely equivalent to* $\{g_v(x)|v \in F_2^n\}$.

Due to Preneel[24], the distribution of absolute Walsh spectra is invariant. We give all functions of same distribution of absolute Walsh spectra a same number $N(walsh(h))$, where $h(x)$ is one of these functions. By Proposition 2, the set $\{N(walsh(f_v(x))|v \in F_2^n\}$ is equal to the set $\{N(walsh(g_v(x))|v \in F_2^n\}$. That is, $M(f) = \{N(walsh(f_v(x))|v \in F_2^n\}$ is invariant. It can be used to classify bent functions. Table 1 lists the number of inequivalent bent functions we get. Except our method, we don't know if there exists other simple method to classify bent functions of trace form.

4.4 On the number of monomials

In 6 variables case, all 3 kinds of inequivalent bent functions are obtained from 2 and 3 monomials. Could all affinely inequivalent 8-variable bent functions be obtained only from 2,3 and 4 monomials? what's the case for more variables?

Table 1. Number of Bent Functions

Bent Number	our algorithm	Fuller's algorithm
8 variables	130	14
10 variables	above 300	46

Or in other words, for any bent function with large linear span, does there exist a bent function with small linear span affinely equivalent to it? If the answer is yes, it would be very promising to know the structure of bent functions in more details because we can only study these bent functions whose linear span is small.

5 Conclusion

The space of boolean functions is very complex, what we know is only a small part. Though there are many research on bent functions, we still don't know it in detail. On this occasion, evolutionary computing is a useful method. Usually it can provide practical solution to problem, what's more, it can lead to theory progress by theoretical analysis of the obtained results.

In this paper, we give an efficient evolutionary algorithm designing bent functions. With it, we obtained many bent functions and made some analysis on them.

First we observe that all 3 affinely inequivalent bent classes in 6-variable can be written as the linear sum of 2 or 3 monomial trace functions. We make a conclusion that affine transform can be used to change the linear span. The conclusion can be used to design bent functions both of large linear span and small linear span and to design perfect nonlinear s-box of non-Niho type.

Second, we find that some exponents take more chances to construct bent functions while some exponents take less. By this observation, we give each exponent a cost function, which make our algorithm more efficient than exhaustive searching algorithm or random algorithm. This is also the advantage over the algorithms based on the algebraic normal form, truth table, or Walsh spectra because we don't know what kinds of algebraic normal form, truth table, Walsh spectra are more possible to be used to construct bent functions.

Third, in order to make a comparison with Fuller's algorithm, we classify them into affinely inequivalent classes. Present experiments show we obtained much more classes than Fuller's algorithm.

References

1. O. S. Rothaus, On "Bent" Functions, J. Combin. Theory Ser. A.,1976, 20, 300-305.
2. R.L.McFarland, A family of noncyclic difference sets, Journal of combinatorics(series A) 15,1-10,1973.

3. J. F. Dillon. Elementary Hadmard Difference Sets. Ph. D, Dissertation, Unv. Maryland, 1974.
4. C. Carlet. Two new classes of bent functions, Advance in cryptology-eurocrypt'93, LNCS765. 77-101, 1994.
5. C. Carlet. Generalized Partial Spreads. IEEE Transacion on I.T., Vol 41, No. 5, 1482-1487, 1995.
6. X. Hou. On the Coefficients of Binary Bent Functions. Proceeding of the American Mathematical Society, Vol. 128, No. 4, 987-996, 1999.
7. E. Bihama, A.Shamir,Differential cryptanalysis of DES-like cryptosystems, Journal of Cryptology, Vol.4, No.1 (1991) 3-72.
8. M. Matsui, Linear cryptanalysis method for DES cipher. LNCS 765, Eurocrypt93, 1994,386-397.
9. K. G. Paterson, Sequences for OFDM and Multi-code CDMA: two problems in algebraic Coding Theory, in T.Helleseth, P.V. Kumar and K.Yang editors, Proceedings of sequence and their applications, SETA01,46-71, Springer-verlag, London, 2002.
10. F. J. Macwillams, J. A. Solane. The Theory of Error-correcting Codes. Northholland Publishing Company, Amsterdam, 1978.
11. J.Fuller,E. Dawson, W. Millan, Evolutionary generation of bent functions for cryptography,Vol. 3, 1655-1661,the 2003 congress on Evolutionary Computation.
12. J.A.Clark,S. Jacob, S. Matria,P. Stanica. Almost boolean functions: the design of boolean fucntions by spectral inversion. Computational Intelligence, Volume 20, Number 3,446-458, 2004
13. Q. Meng, H. Zhang,Z. Wang, et al. Designing bent functions using evolving computing. Acta electronica sinica, 2004, No.11 1901-1903.
14. R.Lidl, H. Niederreiter, finite field, encyclopedia of mathematics and its applications.
15. A.M. Youssef, G.Gong, Hyper-bent functions, Eurocrypto'01, LNCS 2045, 406-419,springer-verlag,2001.
16. K.Nyberg, Perfect non-linear s-boxes, Advances in crypto-eurocrypt'91, LNCS. Vol.547, Springer-verlag,378-386.
17. Y. Niho, Multivalued cross-correlation functions betweeen two maximal linear recursive sequences, Ph.D. Thesis, Univeristy of Southern California. 1972.
18. T.Satoh, T. Iwata,K.kurosawa,on cryptographically secure vectorial boolean functions. Asiacrypt'99,20-28. Springer-verlag.
19. H. Dobbertin, G. Leander, A survey of some recent results on bent functions, Third International ConferenceSequences and Their Applications - SETA 2004,LNCS 3486, Springer 2005,1-29.
20. J.F.Dillon, H. Dobbertin, new cyclic difference sets with singer parameters, finite field and applications, 2004,342-389.
21. J.A. Maiorana, A classification of the cosets of the reed-muller code $R(1,6)$, math. Comp.57,403-414,1991.
22. Fuller. J., Millan. W.. Linear redundancy in S-box. In: Fast Software Encryption, LNCS 2887, Springer-Verlag, 2003, 74-86.
23. Q. Meng, M. Yang, H. Zhang,Y.Liu, Analysis of affinely equivalent boolean functions, The first workshop on boolean functions and application on cryptography, also available at http://eprint.iacr.org, 2005/025.
24. B. Preneel, Analysis and design of cryptographic hash functions, Ph.D thesis, KU Leuven(Belgium),February 1993.

Appendix 1: inequivalent 8-variable bent functions

The 53 affinely inequivalent bent functions of form $tr_1^8(\alpha^i x^{d_1}+\alpha^j x^{d_2}+\alpha^k x^{d_3})$. The parameter (i, d_1, j, d_2, k, d_3) is as follows:

```
0   3   1   5   0  51      0   3  17   9   0  27      0   3   0  15   0  27
1   3  30  39   5  51      0   5   0  15   0  17      0   5   0  15   0  25
2   5  16  15   9  25      2   5  20  15  17  25      2   5  23  15   0  25
0   5   0  15   0  53      3   5   2  25   1  45      3   5  21  25   8  45
0   5   0  45   0  53      8   9   0  17  63  39      0  15   0  17   5  45
0  15   0  17  15  45      0  15   0  17  17  45      0  15   1  17   0  45
3  15   0  17   7  45      5  15   0  17   0  45      0  15   4  25  17  45
0  15  23  25  10  45      1  15  31  25  17  45      7  15   4  25   9  45
7  15  17  25  14  45     10  15  27  25  16  45      4  15  31  27  46  45
3  15   2  27  13  51      0  15  17  39   0  45      4  17   2  19  19  53
0   5  17  15  51  25      1   5   1  15  53  25      1   5  10  15 127  25
1   5  23  15  90  25      1   5  53  15  84  25      1   5  70  15  78  25
1   5 100  15  66  25      1   5 122  15  22  25      1   5 122  15  33  25
2   5  25  15  84  25      2   5  34  15  73  25      2   5  35  15   1  25
2   5  53  15   7  25      3   5  77  15  49  25      3   5  94  15   2  25
4   5   2  15  99  25      0  15  13  25 169  45      1  15  48  25 181  45
1  15  64  25 173  45      3  15  47  25 177  45      3  15  62  25 194  45
7  15  29  25 123  45      7  15  29  25 162  45
```

The 67 affinely inequivalent bent functions of form $f(x) = tr_1^8(\alpha^i x^{d_1}+\alpha^j x^{d_2}+\alpha^k x^{d_3}+\alpha^l x^{d_4})$. The parameter $(i, d_1, j, d_2, k, d_3, l, d_4)$ is as follows:

```
0   5   1  15   5  25 229  45      0   5   1  15  77  25 130  45      0   5   1  15  85  25 155  45
0   5   1  15  85  25 188  45      0   5   3  15  35  25 249  45      0   5   3  15  43  25 207  45
0   5   3  15 170  25  42  45      0   5   5  15 170  25  91  45      0   5   5  15 170  25 181  45
0   5   7  15  85  25 216  45      0   5   7  15 168  25  66  45      0   5   7  15 170  25 161  45
0   5   7  15 186  25 204  45      0   5  11  15 160  25 118  45      0   5  13  15  85  25  26  45
0   5  13  15  85  25 161  45      0   5  13  15  85  25 177  45      0   5  17  15  85  25  68  45
0   5  31  15 170  25 229  45      0   5  43  15 170  25  89  45      1   5   0  15 252  25 110  45
1   5   2  15  96  25 241  45      0   5   1  15   5  25 229  45      0   5   1  15   5  25 229  45
0   5   3  15  35  25 249  45      0   5   3  15  43  25 207  45      0   5   4  15  53  25  10  45
0   5  13  15  10  25  13  45      0   5  22  15  65  25 236  45      1   5   0  15  56  25 155  45
1   5   1  15  56  25  15  45      1   5   2  15  56  25 233  45      1   5   4  15  56  25  19  45
1   5   6  15  56  25  84  45      1   5  11  15  56  25  51  45      1   5  11  15  56  25  74  45
1   5  13  15  56  25 129  45      1   5  16  15  56  25  17  45      1   5  26  15  56  25  33  45
1   5  41  15  56  25 164  45      1   5  56  15  55  25 126  45      1   5  62  15  56  25 143  45
2   5   5  15   9  25 160  45      2   5  18  15   9  25 193  45      3   5   6  15   2  25  27  45
0   5   7  15  33  25   9  85      0   5  27  15  29  25   1  85      0   5  51  15  34  25   1  85
0   5  57  15  19  25   5  85      1   5  14  15   1  25   5  85      1   5  27  15  39  25   5  85
1   5  37  15   5  25   9  85      1   5  37  15  10  25   9  85      1   5  63  15  10  25   1  85
2   5   3  15  16  25   9  85      2   5   3  15  58  25   9  85      2   5   6  15  27  25   5  85
2   5  11  15  38  25   5  85      2   5  41  15  55  25   5  85      3   5   7  15  29  25   9  85
3   5  43  15  19  25   9  85      3   5  43  15  32  25   5  85      3   5  48  15  13  25   1  85
0   5   0  15  51  45   5  85      1   5  17  15  18  45   1  85      1   5  31  15  13  45   9  85
1   5  37  15  60  45   5  85
```

Appendix 2: fast generation of invertible matrix and $f(xA)$

In this appendix, we give an expand algorithm to generate invertible matrix and to calculate the truth table of $f(xA)$, given the truth table of $f(x)$ and the matrix A.

For any given set of size n, like $A = (a_1, a_2, \cdots, a_n)$ there are 2^n kinds of different combinations. If we take $+$ as the combination relationship, then all combinations are

$$0, a_1, a_2, a_1 + a_2, a_3, a_1 + a_3, a_2 + a_3, a_1 + a_2 + a_3, \cdots, a_0 + a_1 + \cdots + a_n.$$

They can be writen into matrix form:

$$\begin{bmatrix} 0\cdots 0\ 0 \\ 0\cdots 0\ 1 \\ 0\cdots 1\ 0 \\ 0\cdots 1\ 1 \\ \cdots\cdots \\ 1\cdots 1\ 1 \end{bmatrix} \times \begin{bmatrix} a_n \\ a_{n-1} \\ a_{n-2} \\ a_{n-3} \\ \cdots\cdot \\ a_1 \end{bmatrix} = \begin{bmatrix} 0 \\ a_1 \\ a_2 \\ a_1 + a_2 \\ \cdots\cdots\cdots\cdots\cdots \\ a_1 + a_2 + \cdots + a_n \end{bmatrix} \tag{4}$$

From the left part of the equation 4, we can give an efficient algorithm.

Expand ALgorithm

input :array $a[]$,the elements of the set, and n, the size of the set.

output: all kinds of 2^n combinations $c[0], c[1], \cdots, c[2^n - 1]$.

```
Expand(int a[],int n, int c[]);
{
 c[0]=0;
 for(i=0;i<n;i++)
    for(j=0;j<(1<<i);j++) c[j+1<<i]=c[j]+a[i];
 }
```

From above algorithm, the computation is

$$1 + \sum_{i=0}^{n-1} 2^i = 2^n.$$

It is optimum in term of computation.

We will use the above algorithm to generate invertible matrix randomly. First we can randomly choose a nonzero element $a_1 \in F_2^n$; Second, we choose the second nonzero element a_2 such that $a_2 \neq a_1$; Thirdly, we choose the third nonzero element a_3 such that a_3 is not in the linear space generated by the elements a_1, a_2; and so on, until we get n elements. We notice that one frequently used computation is generating the linear span from a group of elements. Our expand algorithm can solve this problem.

algorithm: generate invertible matrix randomly

input n,the dimension of the matrix.

output: the row vector $r_1, r_2, \cdots, r_n$ of the generated matrix.

```
int flag[1<<n];
int modu=(1<<n)-1;
int t[1<<n];
for(i=0;i<(1<<n);i++) flag[i]=1;
for(i=0;i<n;i++)
    {
        while(!flag[tmp])  tmp=rand()&modu;
        r[i]=tmp;
```

```
        Expand(r,i+1,t);
        for(j=0;j<(1<<(i+1));j++) flag[t[j]]=0;

    }
}
```

Now we discuss how to get the truth table of $f(xA)$, given the truth table of $f(x)$ and the matrix $A = (r_1, r_2, \cdots, r_n)$, where r_i is the i-th row vector.

Let vector $v = (r_n, r_{n-1}, \cdots, r_1)$, calculate Expand(v,n,c), then the truth table of $g(x) = f(xA)$ can be obtained as follows: $g(x) = f(c[x])$.

In the ordinary method, to calculate $f(xA)$, the computation is $2^n n^2$, while the computation in our method is 2^n. Obviously, much computation is saved by our algorithm. As a basic algorithm, Expand will be used in many areas.