Linearization of Functions Represented as a Set of Disjoint Cubes at the Autocorrelation Domain

Osnat Keren
Bar Ilan University, Israel
kereno@macs.biu.ac.il

Ilya Levin,
Tel-Aviv University, Israel
ilia1@post.tau.ac.il

Radomir S. Stankovic
University of Nis, Serbia
rstankovic@bankerinter.net

Abstract

The implementation cost of a multi-output Boolean function, in terms of the number of two-input AND-OR gates, can be reduced by using a linear decomposition. The linearly decomposed Boolean function consists of a linear function followed by the corresponding linearly transformed function. A complexity of the linearized function and therefore, its implementation cost, depends on the linear transform chosen. In this paper we suggest a spectral technique of the linear transformation of functions defined by disjoint cubes.

The proposed linearization procedure is defined over the autocorrelation domain where the autocorrelation function is represented as an *arithmetic sum of products*. The computation complexity of the suggested method is polynomial in both the number of input variables and the number of cubes of the original function. Hence the suggested method is applicable to functions of a large number of input variables.

Experimental results over standard benchmarks show reduction in the implementation complexity in comparison with the implementation of the initially given non linearized functions. The efficiency in terms of the computation time is demonstrated on randomly generated functions of large number of inputs.

1 Introduction

The problem of linear decomposition of Boolean functions of a large number of input variables is a known and complicated problem. The implementation cost of a function is estimated through the complexity measure for Boolean functions, $\mu(f)$, defined as the number of input pairs at the Hamming distance 1, where the function has identical values. Therefore, the implementation cost ranges from 0 to $n2^n$, where n is the number of variables. Large values of $\mu(f)$ correspond to a simple AND-OR realization while smaller values of $\mu(f)$ correspond to complex realization. The boundary value of $\mu(f) = 0$ is attained by linear Boolean functions.

The linear decomposition proved to be an efficient method for reducing the implementation cost of Boolean functions in terms of the number of two-input gates. The linearly decomposed system consists of a linear function σ followed by a linearized function f_σ, as shown in Figure 1. The linear function σ is implemented by XOR gates, its complexity is polynomial in the number of inputs. The function σ is chosen such that the complexity of implementation of the linearized function f_σ is significantly reduced.

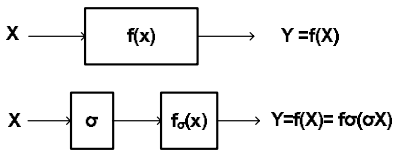

Figure 1: Realization of a function f, (a) direct realization, (b) realization by linear decomposition $f(x) = f_\sigma(\sigma x)$

In 1975 Karpovsky [4] showed a relation between the implementation cost, the measure μ which is the number of adjacent minterms carrying the same output, and the autocorrelation function $R(\tau)$. He showed that the implementation cost is correlated with the sum of the autocorrelation values of the input variables. Hence, by choosing σ that transforms the original input variables to another set of linearly independent variables that carry higher correlation value, the implementation cost of the linearized function can be reduced.

The linear decomposition approach is also used for reducing the size of decision diagrams [3]. In general, there are two methods for deriving the σ and the linearized function $f_\sigma(x)$: the dynamic linearization and the static linearization. The dynamic linearization involves an heuristic search algorithm , e.g. sifting [6]. In some cases, the dynamic approach may fail to find the best set of variables within a given time limitation. The static linearization approach is analytical; a new set of variables is determined by their autocorrelation values. Due to the complexity of finding the optimal solution, greedy algorithms are often used. Nevertheless, the resulting linear function f_σ is on average of a lower implementation cost than the original function.

There are three basic operations involved in the static linearization procedure of a Boolean function $f : GF(2^n) \rightarrow GF(2^k)$ of n input variables and k outputs :

- Calculation of the autocorrelation function $R_f(\tau)$.
- Construction of the linear transform matrix σ, i.e. defining a set of n independent variables (base) having the maximal autocorrelation values.
- Applying a linear transform to the original function.

The autocorrelation function can be calculated in two different ways: a) by definition, and b) by using the Wiener-Khinchin theorem. The efficiency of methods for computing the autocorrelation function depends on the way the function f is specified. In this paper we are interested in functions of a large number of inputs, which are often represented as a set of cubes. Since any set of cubes can be expanded to a set of disjoint cubes, we assume that the function to be linearized is already given as a set of disjoint cubes.

An extensive work has been done in calculations of various discrete spectral transforms, including the Walsh transform of switching functions defined by disjoint cubes, see, for instance, [1, 2] and references therein. However, employing this approach for calculation of the autocorrelation functions R_f by using the Wiener-Khinchin theorem may be inefficient, since the complexity of the method depends on the number of disjoint cubes and the number of different values the Walsh coefficients may take is often large.

Calculation of the autocorrelation of multi-output functions of large number of inputs can be simplified by performing the calculation over the disjoint cubes domain simultaneously on cubes of τ's (see Section 2). In this case the autocorrelation function $R(\tau)$ has a compact representation as an arithmetic sum of products. and the computational complexity of the autocorrelation function is polynomial in the number of inputs and the number of cubes.

A linearization algorithm for efficient minimization of logic functions on the disjoint cubes has been suggested by Varma and Trachtenberg [7] in 1989. The authors reduced the computational complexity by: a) using heuristic algorithm to define a candidate base vector, and b) calculating the autocorrelation function R of the candidate vectors over cubes directly by the definition for the chosen candidate. The main drawback of this method is that the final set of τ's depends on the order of processing the cubes and on the subspace defined by the previously produced base vectors.

Karpovsky, Stankovic and Astola suggested in [3] a linearization algorithm for the reduction of the size of Binary Decision Diagrams (BDD). The corresponding procedure, called the K-procedure, reduces the size of a BDD by performing a linear transformation σ of input variables. The linear transformation is determined as the superposition of linear transformations minimizing the number of nodes per levels, starting from the bottom of the BDD. At each level, the linearization is performed by determining the basis of the inertia group for the function represented by the nodes at the upper levels, and the BDD is folded after each step. The folding of the BDD is an essential step of the K-procedure, it guaranties that the chosen τ is linearly independent from previous τ's. However, the complexity of folding is exponential with the number

of inputs, and therefore the K-procedure may not be applicable to functions of a large number of inputs.

A technique for linearization of multi-output functions of large number of inputs was introduced in [5]. In this method, the complexity of calculation was reduced by performing the calculation entirely over the disjoint cubes. The construction of a linearly independent set of n base vectors is significantly simplified by defining the transform matrix σ as a product of matrices determined by performing a series of instantaneous linear transforms on the set of cubes. The computational complexity of the linearization procedure in [5] is polynomial in the number of inputs n, and the number of cubes, $(\mathcal{O}(n^4 N_{max}^2))$, where N_{max} is the maximal number of products processed per iteration.

Note that a linear transformation of cubes may break a cube into a number of smaller cubes. Since the algorithm in [5] requires to repeat calculation of the autocorrelation per iteration, the complexity of computation may increase from step to step.

In this paper, we suggest an improvement over the algorithm presented in [5]. The suggested technique is a linearization procedure that constructs the transform matrix σ by performing a series of linear transforms directly on the autocorrelation function. This way it is sufficient to calculate the autocorrelation function only once on the initial set of disjoint cubes. As a result, it always derives a linearized function of a lower complexity measure ($\mu(f)$) than the original function.

The paper is organized as follows. In next section 2, we discuss the optimization criterion and the linear decomposition problem. The suggested linearization procedure is presented in Section 3. Section 4 includes simulation results using standard benchmarks and randomly generated Boolean functions. The conclusions summarizing the results are presented in Section 5.

2 Preliminaries

Let $f : GF(2)^n \rightarrow GF(2^k)$ be a system of k logic functions of n variables or multi-output Logic Function. Let $\mathcal{G} = \{0, 1, \phi\}$, where ϕ stands for don't-care. The representation of f at the cubes domain is a set of N pairs

$$F = \{(P_i, Y_i)\}_{i=1}^{N}$$

where $P_i \in \mathcal{G}^n$, is a product and $Y_i \in GF(2^k)$ is the corresponding output.
Two cubes (products) are called disjoint if they do not have any minterm in common. If for a function f, any pair of cubes is disjoint, the function is represented by a disjoint cubes. Clearly, any set of non-disjoint cubes can be expanded into a set of disjoint cubes, and therefore without loss of generality, we assume that a given multi-output Boolean function is specified by a set of N disjoint (orthogonal) cubes (products).
The set of products of a multi-output Boolean function can be partitioned into sets having identical output patterns, called characteristic sets. The characteristic set, F_u, $(u \in GF(2^k),)$ is the set

$$F_u = \{(P_i, Y_i) | (P_i, Y_i) \in F, Y_i = u\} \tag{1}$$

The Boolean function defined by the characteristic set F_u is the characteristic function $f_u(x)$.

2.1 Optimization criterion and the autocorrelation function

As shown in [4], the complexity measure $\mu(f)$ can be written in terms of the autocorrelation function values at τ's of the Hamming weight one; Denote by $R_g(\tau)$ the autocorrelation function of a binary function g, i.e., $R_g(\tau) = \sum_{x \in GF(2^n)} g(x)g(x + \tau)$. Then

$$\mu(f_u) = \sum_{i=0}^{n-1} R_u(\delta_i)$$

where δ_i stands for the representation of 2^i as a binary vector of the length n in base 2, and

$$\mu(f) = \sum_{u \in GF(2^k)} \mu(f_u).$$

At the disjoint cubes domain the calculation of R_u is performed as follows; Let N_u be the number of products $\{P_i\}_{i=1}^{N_u}$ associated with the characteristic set F_u as defined by (1), i.e., $\sum_{u\in GF(2^k)} N_u = N$. Since F is an orthogonal set of products, so does F_u and thus,

$$f_u(x) = \bigcup_{i=1}^{N_u} P_i(x) = \sum_{i=1}^{N_u} P_i(x)$$

where $\bigcup$ stands for OR and $\sum$ is the arithmetic summation. Therefore,

$$R_u(\tau) = \sum_{i=0}^{N_u}\sum_{j=0}^{N_u}\sum_{x\in GF(2^n)} P_i(x)P_j(x+\tau) = \sum_{i=0}^{N_u}\sum_{j=0}^{N_u} R_{i,j}^{(u)}(\tau).$$

To simplify the notations, when it is clear from the context, we omit u, i.e., instead of $R_{i,j}^{(u)}$ we write $R_{i,j}$.

As shown in [5], $R_{i,j}$ can be calculated by determining the coset(cube) of τ's, $C_{i,j} \in \mathcal{G}^n$, for which the product $P_i(x)$ and the shifted product $P_j(x+\tau)$ are not disjoint. Namely, let $P_i = (p_{n-1}^{(i)}, \ldots, p_1^{(i)} p_0^{(i)})$ and $P_j = (p_{n-1}^{(j)}, \ldots, p_1^{(j)} p_0^{(j)}) \in \mathcal{G}^n$, and denote by n_ϕ the size of the set

$$\{k | 0 \le k < n, p_k^{(i)} = p_k^{(j)} = \phi\},$$

the coset determined by the intersection of the product i and the shifted product j is $C_{i,j} = (c_{n-1}, \ldots, c_1, c_0)$, where

$$c_k = \begin{cases} 0 & (p_k^{(i)}, p_k^{(j)}) \in \{(0,0),(1,1)\} \\ 1 & (p_k^{(i)}, p_k^{(j)}) \in \{(0,1),(1,0)\} \\ \phi & otherwise \end{cases}$$

for $0 \le k < n$. The coset is attributed by $V_{i,j} = 2^{n_\phi}$, the number of elements the product $P_i(x)$ and the shifted product $P_j(x)$ have in common. For example, let $P_1 = (01\phi\phi1)$ and $P_2 = (00\phi11)$ then

$$\begin{array}{ll} C_{1,1} = (00\phi\phi0) & V_{1,1} = 2^2 \\ C_{2,2} = (00\phi00) & V_{2,2} = 2^1 \\ C_{1,2} = (01\phi\phi0) & V_{1,2} = 2^1. \end{array}$$

The autocorrelation function can be represented in PLA-like format or equivalently **arithmetic** sum of cubes format. In other words, R can be represented by cubes as a set of $M \le N^2$ pairs,

$$R = \{(C_i, V_i)\}_{i=1}^{M},$$

where $C_i \in \mathcal{G}^n$ is a product and V_i the corresponding integer $1 \le V_i \le 2^n$.

The value of the autocorrelation function $R(\tau)$ is

$$R(\tau) = \sum_{i=1}^{M} C_i(\tau) V_i$$

For example, consider the following 4-input 3-output Boolean function and its total autocorrelation function R, represented as a set of eight cubes

$$F = \left\{ \begin{array}{cc} (0100) & 0 \\ (0011) & 0 \\ \hline (1\phi00) & 1 \\ (0\phi10) & 1 \\ \hline (0101) & 2 \\ (000\phi) & 2 \\ (1\phi1\phi) & 2 \\ \hline (1\phi01) & 3 \\ \hline (0111) & 4 \end{array} \right\} \quad , \quad R = \left\{ \begin{array}{cc} (0000) & 4 \\ (0111) & 2 \\ (0\phi00) & 6 \\ (1\phi10) & 4 \\ (000\phi) & 2 \\ (0\phi0\phi) & 4 \\ (010\phi) & 2 \\ (1\phi1\phi) & 6 \end{array} \right\} \tag{2}$$

The value of the autocorrelation function $R(\tau)$ for $\tau = \delta_3 = (0100)$ is

$$R(\tau) = \sum_{i=1}^{8} C_i(\tau) V_i = 0\cdot 4 + 0\cdot 2 + 1\cdot 6 + 0\cdot 4 + 0\cdot 2 + 1\cdot 4 + 1\cdot 2 + 0\cdot 6 = 12.$$

The complexity measure of this function is

$$\mu = R(0001) + R(0010) + R(0100) + R(1000) = 6 + 0 + 12 + 0 = 18.$$

Next we show how by linear decomposition it is possible to increase μ and hence decrease the implementation cost.

2.2 Linear decomposition

The linear decomposition of a function, presented in [4], allows implementation of f as a superposition of a linear transform function σ implemented by XOR gates followed by a non-linear part, f_σ,

$$f(x) = f_\sigma(\sigma x),$$

of minimal implementation cost as sum of products (see figure 1).

The optimization problem is to find for a given function f, a nonsingular $(n \times n)$ linearization matrix σ, such that $\mu(f_\sigma)$ is maximal.

The autocorrelation functions of $f(x)$ and $f_\sigma(x)$ have the same values but at different positions in the truth-vector, i.e., $R_\sigma(\tau) = R(\sigma^{-1}\tau)$. Therefore, the minimization problem is to determine a nonsingular matrix $\sigma = T^{-1}$, $T = (\tau_{n-1}, \ldots \tau_1, \tau_0)$, such that $\mu(f_\sigma) = \sum_i R(\tau_i)$ is maximal. The columns of T are referred as base vectors that span $GF(2^n)$. Construction of a nonsingular matrix σ, and hence nonsingular T, is equivalent to the problem of construction a set of n base vectors.

In the previous example, the set $\tau_0 = (0100), \tau_1 = (1010), \tau_2 = (0001)$ and $\tau_3 = (0111)$ defines a non singular matrix $T = \sigma^{-1}$ for which the complexity measure of the linearized function f_σ is maximal, i.e.

$$\mu(f_\sigma) = R(T) = 12 + 10 + 6 + 2 = 30.$$

Therefore, the linearized function has lower complexity than the original function. See for example the Karnaugh map of the original and the linearized function given in Table 1.

Table 1: Karnaugh map of f and f_σ

$f(x_3x_2x_1x_0)$

x_3x_2 \ x_1x_0	00	01	11	10
00	2	0	1	1
01	2	2	3	3
11	0	4	2	2
10	1	1	2	2

$f_\sigma(x_3x_2x_1x_0)$

x_3x_2 \ x_1x_0	00	01	11	10
00	2	2	1	4
01	0	2	1	0
11	2	2	1	3
10	2	2	1	3

3 Linearization Algorithm

The complexity of linearization algorithms comes from the calculation of the autocorrelation function and the construction of the set $\{\tau_i\}_{i=0}^{n-1}$. The algorithm presented in [5] performs the linearization over disjoint cubes by an instantaneous linear transformation of the set of cubes. The procedure suggested in this paper performs linearization by instantaneous linear transformation of the autocorrelation function. Both algorithms produce the linear transform matrix σ and a set of linearized disjoint cubes representing f_σ.

The complexity of each iteration of the algorithm in [5] is polynomial in the number of inputs and number of cubes, $(\mathcal{O}(n^w N_i^2))$, where N_i is the number of products at the step i and w is the maximal Hamming weight of τ. However, a linear transformation of cubes may break a cube into a number of cubes of smaller order, therefore the number of products at step i may be larger than N. For this reason, the complexity of the whole linearization procedure of is $\mathcal{O}(n^{w+1} N_{max}^2)$ where the N_{max} is the maximal number of cubes over the steps.

The algorithm suggested in this paper evades from this problem by performing the linear transform on the autocorrelation function, i.e. it exploits the property that the autocorrelation function of the transformed set after applying the instantaneous σ_i is

$$R_i(\tau) = \sum_{x \in GF(2^n)} f_{\sigma_i}(x) f_{\sigma_i}(x + \tau) = \sum_{x \in GF(2^n)} f_{\sigma_i}(\sigma_i x) f_{\sigma_i}(\sigma_i(x + T_i\tau)) = R_{i-1}(T_i\tau) \quad (3)$$

Therefore the autocorrelation of the initial function, denoted by R_0, can be calculated once for $\sum_{j=1}^{w} \binom{n}{j}$ values of τ, and the succeeding autocorrelation functions can be obtained by reordering the elements of R_0 by applying the instantaneous transforms.

Note that calculating the autocorrelation for a restricted set of τ's is equivalent to calculation of the complete R and nullifying the values of R which correspond to τ's of the predefined set. Therefore the autocorrelation function of the transformed set may differ from the transformed partial autocorrelation function.

Linearization procedure by instantaneous linear transforms of R
Set $i = 0$
For all $\tau \in GF(2^n)$, $||\tau|| \leq w$ calculate $R(\tau)$. Set $\sigma = I$
While $i \leq n-1$

1) If $R(\tau) = 0$ for all calculated τ's then break.
2) Determine τ, $\tau \geq 2^{i-1}$ that maximizes $R(\tau)$. In case that there are more than one τ choose one randomly.
3) Construct the instantaneous linear transform matrix σ_i
4) Perform an instantaneous linear transform on R
5) Perform an instantaneous linear transform on the set of products
6) Update σ, $\sigma = \sigma_i \sigma$
7) Increment i

□

Remark 1:
For a function represented as a sum of disjoint cubes, it is more efficient to calculate F_σ in steps where at each step a single base vector is replaced, instead calculating it directly as $F_\sigma = \sigma F$. In other words, the computation complexity is smaller when the linear transform matrix is represented as a product of instantaneous matrices σ_i, namely, denote by F_0 the original set of cubes, and define

$$F_{i+1} = \sigma_i F_i \quad \text{for} \quad i = 0, 1, \ldots n-1$$

then $F_\sigma = F_n$. Note that when a single base vector is replaced, then the instantaneous matrix T_i, $T_i = \sigma_i^{-1}$, is of a special form, i.e., it has $n-1$ columns of Hamming weight one and a single column of the weight greater or equal to one. The corresponding σ_i may transform a cube of the order w to a single cube of the same order or it may break the cube into two cubes of order $w-1$.

Remark 2:
The K-procedure presented in [3] requires a non-singular matrix instantaneous linear transform matrix σ_i for which

$$\sigma\tau = \delta_0,$$

or,

$$\tau = T\delta_0.$$

Namely, the K-procedure restricts only the right most column of T, the other columns can be chosen arbitrarily. Hence, the overall cost function μ after the i'th iteration may be smaller than the original μ. The suggested approach allows to replace a single base vector in each iteration. Since τ carries the maximal autocorrelation value, the vector to be replaced by τ has autocorrelation value smaller than $R(\tau)$. Therefore, the μ cannot decrease from iteration to iteration.

Table 3 shows the values of the transformed autocorrelation of the exemplary function and the chosen τ per iteration. Note that at each step the chosen τ is greater or equal to 2^i. It is easy to see that the autocorrelation value of previously determined base vectors, i.e., $R(\delta_k)$, $k < i$ is decreasing with k. Moreover, the cost function μ is increasing when the chosen τ is a linear combination of **current** base vectors, i.e., when the Hamming weight of τ is one. For example, at the step 1 after replacing the base vector $\delta_1 = (0010)$ by $\tau = (1010)$ the cost function of the transformed set μ_2 $(= 28)$ is greater than the cost function of the current set μ_1 that equals 18. Note however that when the chosen τ is one of the current base vectors, i.e., it is of Hamming weight one, the μ remains unchanged (e.g., $\mu_2 = \mu_3 = 28$).

Table 2: The autocorrelation function and the optimal τ per iteration. Calculation is made without a restriction on the Hamming weight of τ.

i	$R_i(0, 1, \ldots 15)$	τ_i	μ_i
0-original	[16, 6, 0, 0, 12, 6, 0, 2, 0, 0, 10, 6, 0, 0, 10, 6]	4	18
1	[16, 12, 0, 0, 6, 6, 0, 2, 0, 0, 10, 10, 0, 0, 6, 6]	10	18
2	[16, 12, 10, 10, 6, 6, 6, 6, 0, 0, 0, 0, 0, 0, 0, 2]	4	28
3	[16, 12, 10, 10, 6, 6, 6, 6, 0, 0, 0, 0, 0, 0, 0, 2]	15	28
final	[16, 12, 10, 10, 6, 6, 6, 6, 2, 0, 0, 0, 0, 0, 0, 0]	-	30

4 Experimental Results

In this section we provide simulation results on several benchmarks. The performance of the suggested linearization algorithm is examined in terms of the cost function and the execution time. The performance is compared to the original function and to the linearized function after applying K-procedure at the truth-table domain.

Table 3 shows the cost functions for standard benchmarks. The value of the complexity measure $\mu(f)$ for the original function is shown in column denoted by μ. The value μ_{upb} is an upper bound on cost function, it is defined as the sum of the n maximal values of the autocorrelation values. This bound is not always achievable. The μ_{k-proc} stands for the complexity measure obtained by the K-procedure. The values μ_{ALG-I} and μ_{ALG-II} stand for the algorithm presented in [5] and the newly suggested linearization procedure, both obtained with a restriction on the Hamming weight of τ to $w = 3$.

Table 3: μ of the original and linearized benchmark functions and upper bound on μ

Benchmark	n	k	μ	μ_{k-proc}	μ_{ALG-I}	μ_{ALG-II}	μ_{up}
z4	7	4	0	192	320	320	384
rd73	7	3	0	192	384	384	448
inc	7	9	304	304	324	316	464
misex1	8	7	1200	1232	1304	1304	1848
radd	8	5	0	512	704	704	768
dist	8	5	272	316	402	406	496
dc2	8	7	648	648	692	692	948
clip	9	5	384	1032	1448	1400	1736
9sym	9	1	3600	3712	3600	3600	3712
alu2	10	8	2712	2712	2948	2928	3508
dk17	10	11	5950	5950	6006	6006	8762

Figure 2 shows the execution time of the K-procedure at the truth-table domain, the linearization algorithm of [5] and the proposed method with $w = 3$ versus the number of inputs. The execution time was measured on Intel-Centrino, 1.2Ghz, 0.99GB RAM, for random PLA's of four outputs and 50 products. The variance of the measurements was less than 3%. It is clear that linearization at the disjoint cubes domain outperforms linearization based on Wiener-Khinchin theorem.

Table 4 shows the execution time of the linearization procedures with $w = 3$ for random PLAs of 10 to 40 inputs, four outputs and 50 products. As expected, the complexity is polynomial with the number of cubes (N^2), and from the slope of the curves in Figure 2 it is clear that the complexity increases as n^4 with the number of inputs and not exponentially ($n^2 2^n$) as the complexity of the calculating n times the autocorrelation function by the Wiener-Khinchin theorem.

Table 4: Execution-time [sec]of ALG-I and ALG-II vs n for random PLA of 4 outputs and 50 products.

inputs	10	15	20	25	30	35	40
ALG-I	2.64	7.69	21.87	56.38	151.42	339.95	738.03
ALG-II	0.33	0.69	1.55	3.59	8.37	16.97	31.62

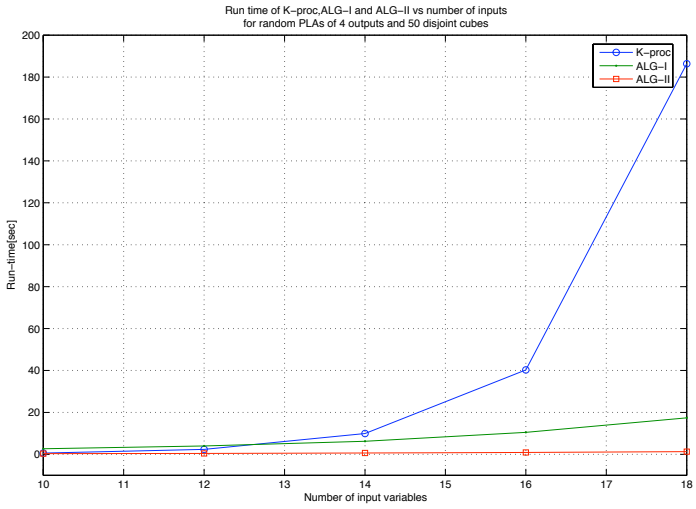

Figure 2: Execution time versus number of inputs of K-procedure and Disjoint cubes linearization algorithms. for random PLA of 4 outputs and 50 products.

5 Conclusion

Linear decomposition has been proven to be an effective tool for reduction of the realization cost of Boolean functions. The present work may be considered as a continuation of the research presented in [7] and [3] for functions having a large number of inputs and defined by disjoint cubes. The main contribution of the paper can be summarized as follows:
1. A method for calculating the autocorrelation function for a logic function defined by its disjoint sum of products is proposed.
2. A technique for simplifying the construction of the set of n independent vectors of high correlation is described, the proposed method represents the transform matrix σ as a product of matrices of a particular form.
The proposed technique is verified over standard benchmark functions and randomly generated Boolean functions for different number of variables and products. The experimental results clearly demonstrate efficiency of the proposed techniques.

References

[1] Falkowski, B.J., Kannurao, S., Circuits and Systems, *ISCAS 2001*. Volume 5, 6-9 May 2001, 61 - 64.

[2] Falkowski, B.J., Schafer, I., Perkowski, M.A., Calculation of the Rademacher-Walsh spectrum from a reduced representation of Boolean functions, *Design Automation Conference, 1992. EURO-VHDL '92, EURO-DAC '92.* 7-10 Sept. 1992, 181 - 186.

[3] M.G. Karpovsky, R.S. Stankovic, J.T. Astola, Reduction of sizes of decision diagrams by autocorrelation functions, *IEEE Trans. on Computers*, Vol. 52, No. 5, 2003, 592-606.

[4] M.G. Karpovsky, *Finite Orthogonal Series in the Design of Digital devices*, New York, Wiley, 1976.

[5] O. Keren and I. Levin, Linearization of the Logic Functions Defined in SOP Form, UROMICRO SEAA / DSD 2005 Porto (Portugal) , 2005 .

[6] D. M. Miller, R. Drechsler and M. A. Thornton, *Spectral Techniques in VLSI CAD*, Kluwer Academic Pub, 2001.

[7] Varma, D. Trachtenberg, E.A., Design automation tools for efficient implementation of logic functions by decomposition; *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Volume 8, Issue 8, Aug. 1989, 901 - 916 .