AND–Decomposition of Boolean Polynomials with Prescribed Shared Variables

Pavel Emelyanov[1,2](✉)

[1] Institute of Informatics Systems, Lavrentiev Ave. 6, Novosibirsk, Russia
emelyanov@iis.nsk.su, emelyanov@mmf.nsu.ru
[2] Novosibirsk State University, Pirogova St. 2, Novosibirsk, Russia

Abstract. In this article, we present an algorithm for conjunctive bi–decomposition of boolean polynomials where decomposition components share only prescribed variables. It is based on the polynomial–time algorithm of disjoint decomposition developed before. Some examples and evaluation of the algorithm are given.

Keywords: AND–decomposition of boolean functions · Combinatorial optimization · Disjoint decomposition · Sharing prescribed variables between decomposition components · Factoring polynomials over finite fields

1 Introduction

Decomposition of boolean functions/formulas is an important research topic having a long history and a wide range of applications including analyses of logic calculi, the theory of games, the (hyper)graph theory, computer algebra algorithms and combinatorial optimization problems. However, boolean function decomposition has attracted the most attention in logic circuit synthesis. It is related to the algorithmic complexity and practical aspects of the implementation of electronic circuits, their size, time delay, and power consumption. Historical and modern issues of decomposition are extensively surveyed in [1,2]. Also, we mention [3–7], which are interesting in the scope of this article.

Bi–decomposition is one of the most important cases of decomposition of boolean functions. Even though it may not be stated explicitly, this case is considered in many papers: [3,5–8], [2, Ch. 3–6]. It has the form $F(X) = \varphi(F_1(\Sigma_1, \Delta), F_2(\Sigma_2, \Delta))$, where $\varphi \in \{\text{OR}, \text{AND}, \text{XOR}\}$, $\Delta \subseteq X$, and $\{\Sigma_1, \Sigma_2\}$ is a partition of the variables $X \setminus \Delta$. Decomposition is called disjoint if $\Delta = \varnothing$. From here on, we will consider conjunctive decomposition only. As an application, we mention solving a variant of the well–known NP–complete Set Splitting Problem known also as the Hypergraph 2–Coloring Problem.

The well–known examples of decompositions are Shannon's Expansions

$$F = xF_{x=1} \vee \bar{x}F_{x=0} = (x \vee F_{x=0})(\bar{x} \vee F_{x=1}),$$

which are powerful tools for theoretical analysis and practical applications. We can establish other decompositions by varying operation bases. For example,

$$F = (xF_{x=0} + x + F_{x=0})\,(xF_{x=1} + x + 1)\,,$$

where + stands for Exclusive–OR. In this paper, this decomposition was deduced as a particular case of a more general decomposition. One disadvantage of these decompositions is that we cannot control the variables sets of their components. Also, if a boolean function F over n variables has $|\mathcal{U}_F|$ units[1] and $|\mathcal{Z}_F|$ zeros, then the number of its conjunctive bi–decompositions equals

$$|\{(G,H) \mid F = G \cdot H\}| = 2^{2^n - |\mathcal{U}_F| - 1} = 2^{|\mathcal{Z}_F| - 1}, \qquad (*)$$

It demonstrates that there exists many of such decompositions but only efficiently computed ones are interesting from a practical point of view.

The authors of [9,10], independently from [11] under more simple settings and in a more simple way, established series algorithms for conjunctive disjoint bi–decomposition for boolean functions represented in Algebraic Normal Form. This form was invented by Zhegalkin [12] and also rediscovered by other researchers. From the algebraic point of view, ANF is a polylinear multivariate polynomial over the finite field of order 2 (Zhegalkin/boolean polynomials, Reed–Muller canonical form, Positive Polarity Form). Hence, conjunctive disjoint decomposition of ANF coincides with factorization of these polynomials (further details in [13]).

In [9,10] it is also demonstrated that these decomposition algorithms for ANF can be straightforwardly transferred to the cases of full DNF and positive DNF. The results are based on the fact that every disjoint decomposable function given in forms DNF (and CNF as well) or ANF uniquely defines the finest partition of its variables. For formulas in CNF/DNF, this follows from the property of a large class of logical calculi shown in [14]. For formulas in ANF, a similar result follows from the fact that the ring of (multivariate) polynomials over the finite field is a unique factorization domain.

In the scope of circuit design, ANF can have some advantages among other representations of boolean functions. For example, it allows for a more compact representation of some classes of boolean functions, e.g. arithmetic schemes, coders, or cyphers. In addition, it has a natural mapping to some circuit technologies (FPGA–based and nanostructure–based electronics) and good testability properties.

Boolean functions in full DNF (i.e. given by explicit enumeration of satisfying vectors) are considered, for example, in the circuit design based on lookup tables (LUTs; see, for example, [15]) because they allow for very efficient operations on table content. Unfortunately, this is space consuming and as such, it bounds number of LUT inputs. Decomposition of a table into smaller tables can enlarge the number of admissible inputs. A potentially interesting application of full DNF decomposition is decomposition of functions in "pre–full" DNF, i.e. whose full DNFs are reconstructed from DNF by a well–known transformation (put $x \vee \bar{x}$ for each missing variable x in monomials), and their sizes increase

[1] $\mathcal{U}_F$ is also called the support of function $supp(F)$. Its cardinality is also called the weight of function $wt(F)$.

reasonably with respect to the original. In the general context of circuit design, for the functions specified by a full DNF, AND–decomposition on the first step of their combinatorial optimization may produce better results due to its "multiplicative" nature. Then, smaller components can be minimized more effectively.

Decomposition of positive boolean functions (monotone functions) given in CNF/DNF attracted particular attention in game theory (simple/voting games) and combinatorial optimization (decomposition of clutters). Please, see the introduction of [5,6] for a summary. In [5,6], Bioch shows that the complexity of AND–decomposition of positive functions is $O(n^5 M)$, where n is the number of variables and M is the number of products in DNF. It follows from the possibility of constructing effectively a representation of all modular sets of a monotone boolean function. A set of variables A is called a modular set of a boolean function $F(X)$ if F can be represented as $F(X) = H(G(A), B)$, where $\{A, B\}$ is a partition of X, and H and G are some boolean functions. The function $G(A)$ is called component of F and a modular decomposition is obtained from iterative decompositions into such components.

Partition of variables is the principal problem in the decomposition of boolean functions. For example, methods described in [4] assume that partitions are supplied. Then they allow us to verify whether a boolean function is decomposable wrt a given variable partition, and to compute its components. The solution, however, implies a number of steps that may be intractable. In [7], the authors propose a graph–theoretical approach. To partition the variable set, the authors describe a procedure to build an undirected "Blocking Edge Graph", where a (minimum) vertex cut determines the partition. The procedure essentially relies on massive checking as to whether some auxiliary boolean functions are equal to zero. Obviously, the efficiency of this step strongly depends on the representation of boolean functions; for some of them this problem can be unfeasible.

Constructing modular sets is a possible way of solving the partition problem for monotone functions in DNF [5,6]. For ANF, a polynomial algorithm finding the bi–partition of variable sets is given in [9,10]. In both cases, once some partition is detected, the components of decomposition can be easily computed. The same ideas are used in [11].

Approaches to boolean function decomposition can be classified into algebraic and logic even though the latter is surely a kind of algebra. In general, logic-based approaches to decomposition are more powerful and achieve better results than the algebraic ones: a boolean function can be decomposable logically, but not algebraically, since boolean factors of a boolean function can differ from its algebraic factors [2, Ch. 4]. A standard algebraic representation of boolean functions is polynomials, usually over finite fields, among which $\mathbb{F}_2$ (the Galois field of order 2) is the best known. Then disjoint AND–decomposition corresponds to factorization/decomposition of multivariate polynomials over $\mathbb{F}_2$ (in general, one distinguishes between decomposition and factorization of polynomials, if they are not multilinear). AND–decomposition of boolean polynomials with shared variables exemplifies finding boolean factors.

The state of the research on the problem of factorization over finite fields is well presented in [13], although it does not contain the key result by Shpilka and

Volkovich [11] reported in 2010. The authors established the strong between the factorization of polynomials over (arbitrary) finite fields and identity testing in these fields. Their results provide that a multilinear polynomial over $\mathbb{F}_2$ can be factored in time $O(L^3)$, where L is the length of the polynomial F given as a symbol sequence, i.e. if the polynomial over n variables has $|F| = M$ monomials of lengths $m_1, \ldots, m_M$ then $L = \sum_{i=1}^{M} m_i = O(nM)$. We also refer $|F|$ as a size of the polynomial. Notice that in [16] these results were extended on polynomials of arbitrary degrees over finite fields and rationals.

In this article, we present an algorithm for conjunctive bi–decomposition of boolean polynomials where decomposition components share only prescribed variables. It is based on the polynomial–time algorithm of disjoint decomposition. Some examples and evaluation of the algorithm are given.

2 $\varnothing$–Decomposition

At first, we briefly outline the algorithm of disjoint decomposition of boolean functions based on variable partition, i.e. a factorization algorithm for multilinear polynomials over $\mathbb{F}_2$, presented in [9,10]. These articles contains the GCD–based decomposition algorithm and the algorithm based on partitioning variable sets. The latter in turn can be implemented either with explicit computation of a product of some polynomials or instead of with multiple evaluations of smaller polynomials.

In the next sections, we assume that the polynomial F does not have trivial divisors of any kinds: neither x nor $x + 1$ divide F. Their interpretation in the scope of decomposition depends on the problem context. We note that besides the factors of the form x and $x + 1$, there is a number of other simple cases of (in)decomposability that can be recognized easily.

We also assume that for F its variable set $Var(F)$ contains at least two variables. $F_{x=v}$ is evaluation of F assuming $x = v$. F'_x represents a (formal) derivative of F with respect to x. Bounding a monomial on a set of variables means removing from monomial all variables that do not belong to this set of variable. The monomial with the empty variable set is 1.

Algorithm of $\varnothing$–Decomposition

```
1. Take an arbitrary variable $x$.
2. Initialize $\Sigma_{same} := \{x\}, \Sigma_{other} := \varnothing$, and $F_{same} := 0, F_{other} := 0$.
3. Compute $G := F_{x=0} \cdot F'_x$.
4. For each variable $y \in Var(F) \setminus \{x\}$
       if $G'_y = 0$ then $\Sigma_{other} := \Sigma_{other} \cup \{y\}$ else $\Sigma_{same} := \Sigma_{same} \cup \{y\}$.
5. If $\Sigma_{other} = \varnothing$ then output $F_{same} := F, F_{other} := 1$ and stop.
6. For each monomial of $F$, bound it on $\Sigma_{same}$ and add this new monomial to
   $F_{same}$ if $F_{same}$ does not contain this monomial.
7. For each monomial of $F$, bound it on $\Sigma_{other}$ and add this new monomial to
   $F_{other}$ if $F_{other}$ does not contain this monomial.
8. Check out which of the products $(F_{same} + c_1)(F_{other} + c_2), c_1, c_2 = 0, 1$, gives
   the original polynomial $F$ and output these components.
```

This algorithm runs in $O(L^3)$ (more precise bounds rely on a careful description of the presentations of polynomials) and is based on identity testing for partial derivatives of a product of polynomials obtained from the input one. Although the algorithm has the same O-complexity as the algorithm of Shpilka and Volkovich, the size of auxiliary data used by the algorithm is smaller, which is significant on large inputs. For instance, the product of polynomials is computed only once, in comparison to the approach described in [11]. In [10] we also show that the algorithm can be implemented without computing the product $F_{x=0} \cdot F'_x$ explicitly, which contributes to the efficiency of the decomposition of large input polynomials.

The following statement provided without a proof quantitatively estimates the evident fact that disjointly decomposable polylinear polynomials are rare.

Proposition 1. *If a random polynomial F has M monomials defined over $n > 2$ variables, then*

$$\mathbb{P}[F \textit{ is } \varnothing\textit{–undecomposable}] > 1 - \left(1 - \frac{\phi(M)}{M}\right)^n > 1 - \left(1 - \frac{1}{e^\gamma \ln\ln M + \frac{3}{\ln\ln M}}\right)^n,$$

where ϕ and γ are Euler's totient function and constant respectively.

3 Δ–Decomposition

Therefore, other kinds of decomposition applicable to a wider class of polynomials are quite interesting. An example of such a generalization is the decomposition where the components share a prescribed set of function variables.

Definition 1. *Δ–Decomposability*
A boolean function F is called AND–decomposable wrt a (possibly empty) subset of variables $\Delta \subseteq \texttt{var}(F)$ (or Δ–decomposable, for short) if it is equivalent to the conjunction $F_1 \wedge F_2$ of some functions F_1 and F_2 such that

1. $\texttt{var}(\psi_1) \cup \texttt{var}(\psi_2) = \texttt{var}(\varphi)$;
2. $\texttt{var}(\psi_1) \cap \texttt{var}(\psi_2) \subseteq \Delta$; *and*
3. $\texttt{var}(\psi_i) \setminus \Delta \neq \varnothing$, *for* $i = 1, 2$.

The functions F_1 and F_2 are called Δ–decomposition components of F. We say that F is Δ–decomposable with a variable partition $\{\Sigma_1, \Sigma_2\}$ if F has some Δ-decomposition components F_1 and F_2 over the variables $\Sigma_1 \cup \Delta$ and $\Sigma_2 \cup \Delta$, respectively.

The following function has no disjoint decomposition but it has $\{x\}$–decomposition:

$$x + ux + vx + uvx + ust + vst + stx + uvstx = (x + u + v + xuv)(x + st)$$

The following function has no disjoint decomposition and any single shared variable decomposition but it has decomposition with two shared variables

$$ytuv + stuv + suvx + yst + ysx + ytx + stx + yt + sx = (xs + yt + st)(x + y + uv).$$

The case $\Delta = \mathtt{var}(F)$ seems trivial because such decomposition obviously exists for every boolean function F. As well the statement $(*)$ from **Introduction** tells us that there exists a lot of AND–decomposition. Probably from the circuit design point of view the following decomposition

$$uvx + uvy + uxy + vxy = (u + v + x + y)(uv + ux + uy + vx + vy + xy)$$

is not appropriate. However, effective finding decompositions with low degree components or good structural properties could be very useful for cryptanalytic purposes.

Notice that the Shannon-like decomposition mentioned in **Introduction** $F = (xF_{x=0} + x + F_{x=0})(xF_{x=1} + x + 1)$ is a $\{x\}$–decomposition of F if $\mathtt{var}(F_{x=0}) \cap \mathtt{var}(F_{x=1}) = \varnothing$. Because $F'_x = F_{x=0} + F_{x=1}$ for multilinear polynomials it follows that

$$F(U, V, x) = (G(U) + H(V))x + G(U) = xH(V) + (x + 1)G(U).$$

The function

$$\begin{aligned} F = {} & stuv + stux + stvy + stxy + suvx + svxy + tuvy + tuxy + uvxy + \\ & sux + sxy + tvy + txy + uxy + vxy + xy \end{aligned}$$

has both disjoint decomposition and $\{x, y\}$–decomposition

$$F = (tv + tx + vx + x)(su + sy + uy + y) = (uv + ux + vy + xy)(st + sx + ty + xy).$$

The reason is that this function has finer decomposition

$$F = (v + x)(t + x)(u + y)(s + y)$$

admitting different combinations for the bi–decompositions. Quite interesting that the components of $\varnothing$–decomposition are irreducible over $\mathbb{F}_2$ in contrast with $\{x, y\}$–decomposition where the components can be further decomposed.

Finally, the function

$$F = ((x + y)(u + v)(p + q) + (xy + 1)(uv + 1)(pq + 1))s + (x + y)(u + v)(p + q)$$

provides an example having three $\{s\}$–decompositions; the reader can easily reconstruct them.

At first, we consider some decomposition which does not guarantee Δ–disjointness of components but elucidates some details.

3.1 “Δ–unpredictable” Decomposition

The decomposition algorithm under development relies on solving the equation $XY + DX + EY + F = 0$ over boolean polynomials. The idea comes from the algorithmics of diophantine quadratic hyperbolic equations. The sequence of transformations

$$\begin{aligned} xy + dx + ey &= f \\ xy + dx + ey + de &= f + de \\ (x + e)(y + d) &= f + de \end{aligned}$$

leads us to two cases:

– $f + de = 0$. Then the following two solutions are possible:
  - $x = -e$ and an arbitrary y; and
  - $y = -d$ and an arbitrary x.

  Notice that this case is impossible for boolean polynomials because the polynomials of interest have no trivial divisors.
– $f + de \neq 0$ and $f_1 \cdot f_2 = f + de$. Then the following two solutions are possible:

$$\begin{cases} x = f_1 - e \\ y = f_2 - d \end{cases} \quad \text{or} \quad \begin{cases} x = f_2 - e \\ y = f_1 - d. \end{cases}$$

Let us return to the boolean polynomial equations. If decomposition exists wrt some variable, then the following identities hold

$$(A_x x + A_\varnothing)(B_x x + B_\varnothing) = (A_x B_x + A_x B_\varnothing + A_\varnothing B_x)x + A_\varnothing B_\varnothing = xF'_x + F_{x=0}.$$

$$\begin{cases} A_\varnothing B_\varnothing = F_{x=0} \\ A_x B_x + A_x B_\varnothing + A_\varnothing B_x = F'_x. \end{cases}$$

Taking into account $F'_x + A_\varnothing B_\varnothing = F'_x + F_{x=0} = F_{x=1}$, we get

$$(A_x + A_\varnothing)(B_x + B_\varnothing) = F_{x=1}.$$

Going over all possible disjoint decompositions $F_{x=0} = A_\varnothing B_\varnothing$ and $F_{x=1} = f_1 f_2$, we finally arrive at:

$$\begin{cases} A_x = f_1 + A_\varnothing \\ B_x = f_2 + B_\varnothing \end{cases} \quad \text{or} \quad \begin{cases} A_x = f_2 + A_\varnothing \\ B_x = f_1 + B_\varnothing. \end{cases}$$

In particular, we can choose $A_\varnothing = 1$, $B_\varnothing = F_{x=0}$, $f_1 = F_{x=1}$, $f_2 = 1$, and it yields the Shannon–like expansion $F = (xF_{x=0} + x + F_{x=0})(xF_{x=1} + x + 1)$ mentioned above. A simple corollary of this expansion is

$$F = 1 \quad \Longleftrightarrow \quad \begin{cases} (x+1)(F_{x=0} + 1) = 0 \\ x\,(F_{x=1} + 1) = 0, \end{cases}$$

which suggests an idea of a polynomial–time SAT–ANF algorithm.

Let us briefly review the case $|\Delta| = 2$. Given an F and variables x, y. Then

$$\begin{aligned} F &= xyF''_{xy} + x(F_{y=0})'_x + y(F_{x=0})'_y + F_{x=0,y=0} \\ &= (A_{x,y}xy + A_x x + A_y y + A_\varnothing)(B_{x,y}xy + B_x x + B_y y + B_\varnothing). \end{aligned}$$

Expanding, simplifying, and equaling correspondent coefficients we have the following system of polynomial equations:

$$\begin{cases} A_\varnothing B_\varnothing = F_{x=0,y=0} \\ A_x B_x + A_x B_\varnothing + A_\varnothing B_x = (F_{y=0})'_x \\ A_y B_y + A_y B_\varnothing + A_\varnothing B_y = (F_{x=0})'_y \\ A_{x,y}B_{x,y} + A_{x,y}(B_x + B_y + B_\varnothing) + B_{x,y}(A_x + A_y + A_\varnothing) + A_x B_y + A_y B_x = F''_{xy}, \end{cases}$$

we can proceed analogously to the case $|\Delta| = 1$.

3.2 Decompositions with Non–empty Prescribed Δ

Let F be a boolean polynomial over the variables $\mathtt{var}(F) = \{x_1, \ldots, x_n\}$ and $\Delta \subseteq \mathtt{var}(F)$, $|\Delta| = k$, be a set of shared variables of the bi–decomposition we are trying to find. Every monomial $\prod_{x_i \in \delta} x_i$, $\delta \subseteq \Delta$, including $\varnothing$, has the coefficients A_δ and B_δ in the corresponding components of the bi–decomposition. These coefficients, which are polynomials over the variables $\mathtt{var}(F) \setminus \Delta$ satisfy the following system of 2^k equations:

$$\text{for all } \delta \subseteq \Delta \qquad \sum_{\substack{\forall \alpha,\beta \subseteq \delta \\ \alpha \cup \beta = \delta}} A_\alpha B_\beta = F|_\delta \,, \quad \text{where} \quad F|_\delta = (F|_{x=0, x \in \Delta \setminus \delta})|'_{y, y \in \delta} \,.$$

As previously noted, solving this system starts with finding all disjoint decompositions (i.e. $\mathtt{var}(A_\varnothing) \cap \mathtt{var}(B_\varnothing) = \varnothing$):

$$A_\varnothing B_\varnothing = F|_\varnothing \,.$$

Propagating these and subsequently found decompositions we can deduce all solutions of the system. Because we are interested in Δ–decompositions, we have to maintain the disjointness of variables sets $\cup_\delta \mathtt{var}(A_\delta)$ and $\cup_\delta \mathtt{var}(B_\delta)$.

To estimate the algorithm's time complexity, we would make some preliminary remarks. [9,10] describe cubic algorithms for the disjoint bi–decomposition of boolean polynomials. Recall that the basic idea of one of them is to partition the variable set into two sets (if exists) with respect to one selected variable:

– one of them contains this variable and corresponds to an undecomposable component of decomposition; and
– another one corresponds to the second component that might be further decomposable.

Then, these decomposition components can be easily reconstructed.

It is important to note that the selected variable must not be a trivial divisor of the polynomial of interest. If the polynomial has t trivial divisors, then it has at least 2^{t-1} disjoint bi–decompositions corresponding to the bi–partitions of this set of trivial divisors. It follows that every boolean polynomial over n variables has at most $d = \max(n, 2^{t-1}(n - t))$ disjoint bi–decompositions, and this bound can be improved under additional conditions.

Hence, precise estimation of the worst–time complexity of Δ–decompositions is quite difficult in the presence of trivial divisors for intermediate decompositions. An upper bound for this multiplier can be $O(n^k)$ but it is quite coarse. We give the worst–time complexity under the assumption that all intermediate decompositions produce two components.

Estimation of the worst–time complexity of the Δ–decomposition algorithm involves the estimation of complexity of solving 2^k equations which includes

– varying decompositions of the previous steps that can produce several versions of each equation; the number of versions for the last equation can be bounded as 2^k;

- coefficients of each next equation are computed with the help of the solutions of the previous equations; the complexity can be estimated as $kS(k,n,M)$, where $S(k,n,M)$ is complexity of the summation of k boolean polynomials with at most n variables and at most M monomials; and
- for each equation, disjoint decomposition needs to be done; let its complexity be $T(n,M)$.

Putting all together, we have $O(k2^{2k}S(k,n,M)T(n,M))$.

We can make an important observation affecting the algorithm's complexity. If the variable set of $F|_\varnothing$ contains all variables of F outside $\varDelta$, i.e. $\mathtt{var}(F|_\varnothing) = \mathtt{var}(F) \setminus \varDelta$, and we deduce some decomposition $A_\varnothing B_\varnothing = F|_\varnothing$, then all subsequent decompositions can avoid the step of the partition of variables sets because it has been already determined by the couple $\mathtt{var}(A_\varnothing)$ and $\mathtt{var}(B_\varnothing)$. This can reduce time complexity from cubic to quadratic with respect to the lengths of polynomials. Even if not all variables of F outside $\varDelta$ appear in $F|_\varnothing$, we can check only these variables with respect to one part of the partition to complete the decomposition.

3.3 Examples and Experimental Evaluation

From the circuit design point of view, optimization quality of decompositions is essential. In contrast with disjoint decomposition when the sizes of components are always less than the size of the original polynomial, decompositions with shared variables have components, sizes of which can vary in wide range. We consider the case $|\varDelta| = 1$: $F(X,Y,s) = F_1(X,s)F_2(Y,s)$, $X \cap Y = \varnothing$. The decomposition quality is the ratio

$$Q_F = \frac{|F|}{|F_1| + |F_2|}.$$

It is easy to construct a family of boolean functions over $n \geq 5$ variables such that for every function F from its decomposition quality is

$$Q_F = \begin{cases} 2^{\frac{n-3}{2}}, n & \text{is odd,} \\ \frac{1}{5}2^{\frac{n+2}{2}}, n & \text{is even.} \end{cases}$$

These functions base on "bi–partitions" of the set of all monomials over $n-1$ variables: sums of every subset and its complement[2] form respectively a derivative and 0–evaluation of the function of interest wrt a shared variable of decomposition. Decomposition components of these functions have the same construction. A 5–variable boolean function belonging to this family is

$$\begin{aligned}(s + x_1 + x_2 + x_1x_2)(s + y_1 + y_2 + y_1y_2) = \\ s + sx_1 + sx_2 + sy_1 + sy_2 + x_1y_1 + x_1y_2 + x_2y_1 + x_2y_2 + \\ sx_1x_2 + sy_1y_2 + x_1y_1y_2 + x_2y_1y_2 + x_1x_2y_1 + x_1x_2y_2 + x_1x_2y_1y_2\end{aligned}$$

[2] To exclude polynomials with trivial divisors they have to be relative prime.

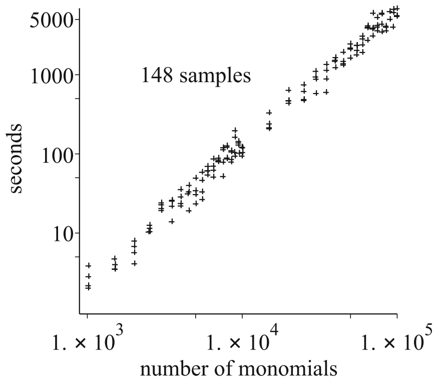

Fig. 1. Polynomial with 100 variables

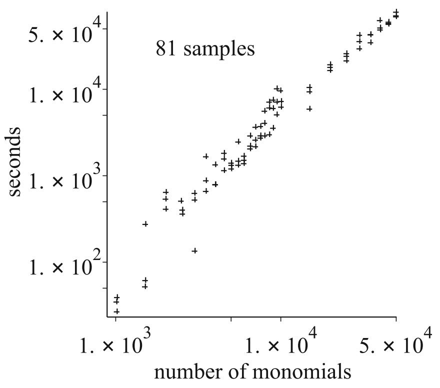

Fig. 2. Polynomial with 1000 variables

The total sizes of decomposition components can be also larger than the size of the original polynomial. 3–variables function examples are

$$\begin{aligned} sx_1y_1 + s + 1 &= (sx_1 + s + 1)(sy_1 + s + 1) \\ sx_1 + sy_1 + x_1 &= (sx_1 + s + x_1)(sy_1 + s + 1) \\ sx_1y_1 + x_1y_1 + s &= (sx_1 + s + x_1)(sy_1 + s + y_1) \end{aligned}$$

and a 5–variables example is

$$sx_1y_2 + sx_2y_1 + x_1y_2 = (sx_1 + sx_2 + x_1)(sy_1 + sy_2 + y_2).$$

For all these functions the decomposition quality $Q = \frac{1}{2}$, i.e. we observe regression of decomposition representation instead of its improvement.

For computational evaluation of the developed decomposition algorithm, we use Maple 17 for Windows run on 1.6 GHz notebook with 12 GB RAM. The figures show the plots of decomposition time depending on the number of monomials of random polynomials. These polynomials have 100 or 1000 variables, surely containing two decomposition components of almost equal sizes, sharing one variable (Figs. 1 and 2).

4 Final Remarks

Not only cases of decomposition with the prescribed Δ (empty or non) can be interesting. Some other cases of interest are:

– The "pure" product can be spoiled by a few monomials

$$F(X,Y) = G(X)H(Y) + D(X,Y), \quad \text{where} \quad |F| \gg |D|,$$

i.e. the "defect" $D(X,Y)$ can extend or shrink this product. Its detecting allows us to provide a more compact form of the original polynomial.

– A set of shared variables Δ can be *à priori* unknown. It is computed in the course of the algorithm, and the induced decompositions should fit different optimality criteria which can involve, for instance among others, minimum Δ or Δ such that components of decomposition are as balanced as possible.

The last generalization attracts attention to the problem how Δ–decomposability depends on cardinality of Δ. Is it possible to estimate probability $\mathbb{P}(n, |F|, |\Delta|)$ of that we can decompose, a polynomial F over n variables among which the subset Δ is large enough?

Since decompositions with shared variables can have components, sizes of which vary in wide range, it is interesting to estimate the average decomposition quality over all boolean functions with n–variables.

As it is mentioned in **Introduction** positive boolean functions play an important role in the combinatorial optimization and graph/game theory. We know that the algorithm of disjoint decomposition of boolean functions in ANF can be transferred on boolean functions in positive and full DNF. Is it possible to do the same for Δ-decomposition?

References

1. Perkowski, M.A., Grygiel, S.: A survey of literature on function decomposition, Version IV. PSU Electrical Engineering Department Report, Portland State University, Portland, Oregon, USA, November 1995
2. Khatri, S.P., Gulati, K. (eds.): Advanced Techniques in Logic Synthesis, Optimizations and Applications. Springer, New York (2011)
3. Mishchenko, A., Sasao, T.: Large-scale SOP minimization using decomposition and functional properties. In: Proceedings of the 40th ACM/IEEE Design Automation Conference (DAC '03), pp. 149–154. ACM, New York (2003)
4. Steinbach, B., Lang, C.: Exploiting functional properties of Boolean functions for optimal multi-level design by bi-decomposition. Artif. Intell. Rev. **20**(3–4), 319–360 (2003)
5. Bioch, J.C.: The complexity of modular decomposition of Boolean functions. Discrete Appl. Math. **149**(1–3), 1–13 (2005)
6. Bioch, J.C.: Decomposition of Boolean functions. In: Crama, Y., Hammer, P.L. (eds.) Boolean Models and Methods in Mathematics, Computer Science, and Engineering. Encyclopedia of Mathematics and its Applications, vol. 134, pp. 39–78. Cambridge University Press, New York (2010)
7. Choudhury, M., Mohanram, K.: Bi-decomposition of large Boolean functions using Blocking Edge Graphs. In: Proceedings of the 2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '10), pp. 586–591. IEEE Press, Piscataway (2010)
8. Mishchenko, A., Steinbach, B., Perkowski, M.A.: An algorithm for bi-decomposition of logic functions. In: Proceedings of the 38th ACM/IEEE Design Automation Conference (DAC '01), pp. 103–108. ACM, New York (2001)
9. Emelyanov, P., Ponomaryov, D.: On tractability of disjoint AND-decomposition of Boolean formulas. In: Voronkov, A., Virbitskaite, I. (eds.) PSI 2014. LNCS, vol. 8974, pp. 92–101. Springer, Heidelberg (2015)

10. Emelyanov, P., Ponomaryov, D.: Algorithmic issues of conjunctive decomposition of boolean formulas. Programming and Computer Software 41(3) (2015) 162–169 Translated: Programmirovanie, vol. 41, No. 3, pp. 62–72 (2015)
11. Shpilka, A., Volkovich, I.: On the relation between polynomial identity testing and finding variable disjoint factors. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 408–419. Springer, Heidelberg (2010)
12. Zhegalkin, I.: Arithmetization of symbolic logics. Sb. Math. **35**(1), 311–377 (1928). In Russian
13. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra, 3rd edn. Cambridge University Press, New York (2013)
14. Ponomaryov, D.: On decomposability in logical calculi. Bull. Novosibirsk Comput. Cent. **28**, 111–120 (2008)
15. Kuon, I., Tessier, R., Rose, J.: FPGA Architecture: Survey and Challenges. Now Publishers Inc., Boston - Delft (2008)
16. Kopparty, S., Saraf, S., Shpilka, A.: Equivalence of polynomial identity testing and polynomial factorization. Comput. Complex. **24**(2), 295–331 (2015)