Genetic algorithm for minimization of ESOP representations for multiple-output logic functions

A Kazimirov and V Maleyev
Institute of Mathematics and Information Technologies, Irkutsk State University, 1 K. Marks St., 664003 Irkutsk, Russia
E-mail: a.kazimirov@gmail.com

Abstract. This article concerns ESOP representations for multiple-output logic functions. Exclusive-or sum-of-products expression (ESOP) is a formula representing a logic function as an exclusive-or sum of products of variables and their negations. EXOR-based realization leads to reduction in size of logic circuits for many practical logic functions and improves their testability. Minimization of a logic function is a problem of constructing ESOP representation for the given function having the minimal number of summands. We introduce a correspondence between ESOPs for multiple-output logic functions and single-output logic functions and propose genetic algorithm obtaining close to minimal ESOP representations for multiple-output logic functions.

1. Introduction
Polynomial or exclusive-or sum-of-products expression (ESOP) is a representation of a logic function as an exclusive-or sum

\[ f(x_1, \ldots, x_n) = K_1 \oplus \cdots \oplus K_s, \]

where \( K_i \) is a product of variables and their negations.

Among the classes of AND-EXOR expressions, ESOPs are the most general AND-EXOR expressions and require the fewest product terms to represent logic functions. Logic circuits including EXOR gates have some advantages over circuits with only AND and OR gates. EXOR-based realization can reduce circuit size for many practical logic functions [1] and improve their testability [2]. Any logic function of \( n \) arguments can be represented as \( 2^{3n-2^n} \) different ESOPs. These representations may vary in their size. Complexity of an ESOP is a number of summands in it. Complexity of a logic function \( f \) is defined as the size of the minimal ESOP of \( f \) and is denoted as \( L(f) \).

Complexity for all \( n \)-ary functions \( L(n) \) is defined as the complexity of the most complex \( n \)-ary function:

\[ L(n) = \max_{f \in F_n} L(f). \]

Minimization of a logic function is a problem of obtaining the minimal ESOP representation for given function.

Size of a formula representing given logic function influences on size and performance of logical circuits built on this formula.
A lot of algorithms for minimization of ESOPs by applying heuristic rewriting rules have been proposed. However, these algorithms do not guarantee the minimality of resulting ESOPs. Although an algorithm based on a function decomposition \([3, 4, 5]\) computes a minimal ESOP of a given function \(f\) of \(n\) arguments by exhaustive search through all \((n - 1)\)-ary functions, this algorithm requires a huge amount of time for \(n \leq 6\).

Some minimization algorithms obtaining minimal ESOPs or representations in their subclasses \([6, 7]\) use genetic and evolutionary models to reduce search space \([8, 9]\). This paper considers modification of the genetic algorithm introduced in \([13]\).

2. Multiple-output logic functions

We will refer to logic functions as single-output functions opposed to multiple-output functions.

A multiple-output logic functions is a mapping

\[ F : \{0, 1\}^n \rightarrow \{0, 1\}^k. \]

Thus \(F\) can be considered as a set of \(k\) functions of \(n\) arguments:

\[ S = \{f_1, \ldots, f_k\}. \]

Simultaneous realization of these functions can take advantage in reusing already implemented summands. Thus complexity of multiple-output function \(S\) is defined as the minimal number of distinct summands in all ESOPs:

\[ L(S) = \min_{P_1, \ldots, P_k} \left\lvert \bigcup_{i=1}^k M(P_i) \right\rvert, \]

where \(P_i\) is an ESOP for \(f_i\) and \(M(P_i)\) is a set of all summands in \(P_i\).

Denote the maximal complexity of \(n\)-input \(k\)-output functions as \(L(n, k)\). The trivial upper bound on complexity gives \(L(n, k) \leq k \cdot L(n)\).

**Statement.** \(L(n, 2^k) \leq L(n + k)\).

Given \(n\)-input \(2^k\)-output function \(S = \{f_1, \ldots, f_{2^k}\}\) we can construct a logic function \(g\) with auxiliary variables \(y_1, \ldots, y_k\), such that \(g(x_1, \ldots, x_n, \alpha_1, \ldots, \alpha_k) = f_m(x_1, \ldots, x_n), \) where \(m = 2^{k-1} \alpha_1 + 2^{k-2} \alpha_2 + \ldots + \alpha_k + 1\).

For every ESOP \(\Phi\) for \(g\), we can construct corresponding ESOPs for \(f_i\) by substituting values for \(y_1, \ldots, y_k\) equal to bits in binary code of \((i - 1)\).

So we have \(L(\Phi)\) summands representing every function in \(S\) and \(L(S) \leq L(\Phi)\). If \(\Phi\) is the minimal ESOP for \(g\) we have \(L(\Phi) = L(g)\).

Thus \(L(S) \leq L(g)\). By definition \(L(g) \leq L(n + k)\), so \(L(S) \leq L(n + k)\), and finally \(L(n, k) \leq L(n + k)\).

It is shown in \([13]\) that \(L(n, 2) = L(n + 1)\).

With this approach we can construct minimization algorithm for multiple-output functions by reduction to single-output functions.

3. Function decomposition

One of the approaches for logic function minimization relies on a decomposition of logic function to the sum of three functions having fewer number of arguments.

Every ESOP representation of a logic function \(f(x_1, \ldots, x_n)\) can be rewritten as following:

\[ x_n f_1(x_1, \ldots, x_{n-1}) \oplus \bar{x}_n f_2(x_1, \ldots, x_{n-1}) \oplus f_3(x_1, \ldots, x_{n-1}). \]

This decomposition can be used for minimization since it represents \(n\)-ary function with functions having smaller arity. These 3 functions also can be represented as ESOPs with no
occurrences of $x_n$ or its negation. In other words, $x_nf_1$ is a part of the ESOP of $f$ which contains $x_n$; $\bar{x}_nf_2$ is a part of the ESOP of $f$, which contains negation of $x_n$; $f_3$ is a part of the ESOP of $f$, which does not contain any occurrence of $x_n$.

If functions $f_1$, $f_2$ and $f_3$ are chosen in such a way that sum of their complexities would be minimal then it is possible to build minimal ESOP for $f$:

$$L(f) = \min (L(f_1) + L(f_2) + L(f_3)).$$

To minimize function $f$ with this equation one have to look through all $f_1$, $f_2$, $f_3$ functions. Among these functions only one can be arbitrary chosen, others are uniquely defined by the next equations:

$$f(x_1, ..., x_{n-1}, 0) = f_2(x_1, ..., x_{n-1}) \oplus f_3(x_1, ..., x_{n-1}),$$

$$f(x_1, ..., x_{n-1}, 1) = f_1(x_1, ..., x_{n-1}) \oplus f_3(x_1, ..., x_{n-1}).$$

Any of these functions could be chosen as parameter, others would be defined by this one. Thus minimization of an $n$-variable function $f$ can be reduced to a problem of finding such $(n - 1)$-ary function $f_3$ that the value $L(f_1) + L(f_2) + L(f_3)$ would be minimal.

For decreasing the depth and number of iterations it is possible to use equivalence classes by transformations that preserve complexity of a ESOP. The most general of these transformations are LP-transformations [10, 11, 12]. LP-transformation of a formula is a permutation of its variables and/or permutation of summands $f_1$, $f_2$, $f_3$. These transformations do not change number of summands in the ESOP, so LP-equivalent functions have the same complexity.

Exact minimization of logic functions is possible only for $n \leq 6$ [3] and for particular functions of more variables [4]. This algorithm is based on a search of a complexity of an $n$-variable function by reduction to an exhaustive search through all $(n - 1)$-ary functions with precalculated complexities. For a function having $n \leq 4$ arguments it is possible to calculate complexity by exhaustive search with reduction to $n = 1$. For bigger number of variables it is useful to store precalculated complexities for functions with a fewer number of variables.

4. Genetic algorithm for single-output functions minimization

Different restrictions on a search space for function $f_3$ can lead to different algorithms of approximate minimization. Such algorithms will give an upper bound on the ESOP’s complexity and an ESOP representation which for some functions would be worse than minimal one.

One approach to reduce the search space is to use a genetic model.

A logic function could be defined in different ways. The most common way is to define it by formula. Since the domain of the function is finite it is possible to define this function by a table of all its values in the lexicographical order of the arguments vector. In this case arguments’ values are redundant and function could be stored just as a vector of its values. So, for example, function $x_1 \& x_2$ could be represented as a vector $(0001)$.

Genetic algorithm for Boolean function minimization is based on using the vector of the function $f_3$ as a chromosome for an individual of a population. So the algorithm could be represented as follows.

Create starting population of possible candidates for the function $f_3$. There are different ways to generate such functions but we suppose that they are generated as pseudo-random bit sequences. Every function in the population is evaluated by a fitness function which is equal to $L(f_1) + L(f_2) + L(f_3)$. The population is sorted by a value of the fitness functions in the descending order.

After this individuals from the population are selected for mutation and crossover with probability linearly decreasing from first to last individuals.
Mutation is implemented as a random bit inversion in a chromosome. Crossover is implemented by combining elements of two chromosomes corresponding to subfunctions with \( x_i = 0 \) and \( x_i = 1 \).

**Proposition.** One bit change in the vector of the function \( f_3 \) changes the complexity of the corresponding ESOP for \( f \) no more than by 3.

Thus the search space is smooth which implies a good applicability of the genetic model for the minimization problem. In [13] an upper bound on complexity of all 7-variable functions is presented. This bound is calculated via minimization of certain function classes with genetic algorithm.

5. Genetic algorithm for multiple-output functions minimization

Consider \( n \)-input \( k \)-output function \( S = \{ f_1, \ldots, f_k \} \).

We use \( k \) sets \( \{ M_1, \ldots, M_k \} \) of different conjunctions of \( x_1, \ldots, x_n \) and their negations as a chromosome. Every set \( M_i \) represents shared summands for \( f_i \). Thus the set of all shared summands is equal to \( \bigcup_{i=1}^{k} M_i \). And we have to find ESOPs for remaining functions \( f_i \oplus P(M_i) \), where \( P(M_i) \) is an EXOR sum of all elements in \( M_i \).

We define fitness function as following:

\[
J(\{M_1, \ldots, M_k\}) = \left| \bigcup_{i=1}^{k} M_i \right| + \sum_{i=1}^{k} L(f_i \oplus P(M_i)).
\]

Thus the fitness function gives an upper bound on the ESOP complexity for \( S \), and the minimization is reduced to a search for a minimum of \( J \) through all sets of summands.

ESOPs for intermediate logic functions are found by minimization algorithm for single-output functions.

Mutation is implemented as a random insertion of deletion of a summand in any set. Crossover is implemented as a random summands exchange between corresponding sets for two chromosomes.

The proposed algorithm was implemented and tested on 1000 randomly generated 6-input 2-output and 6-input 3-output functions.

For 6-input 2-output functions the proposed algorithm obtains ESOPs on average having 0.7 more summands than the exact minimization of two corresponding single-output functions.

For 6-input 3-output functions the genetic algorithm obtains ESOPs on average having 2.2 fewer summand than the exact minimization of three corresponding single-output functions.

References

[1] Debnath D and Sasao T 1998 A heuristic algorithm to design AND-OR-EXOR three-level networks Proc. Asia and South Pacific Design Automation Conf. 6974
[2] Chatterjee M, Pradhan D K and Kunz W 1998 LOT: Logic Optimization with Testability. New Transformations for Logic Synthesis IEEE Trans. CAD 17 5 38699
[3] Gaidukov A 2002 Algorithm to derive minimum ESOPs for 6-variable functions Proc. 5th Int. Workshop on Boolean Problems 141-8
[4] Hirayama T and Nishitani Y 2009 Exact minimization of AND-EXOR expressions of practical benchmark functions J. of Circuits, Systems and Computers 18 46586
[5] Nishitani Y and Shimizu K 1994 Lower bounds on size of periodic functions in exclusiveOR sum-of-products expressions IEICE Trans. Fundamentals, E77-A 47582
[6] Vinokurov S F, Ryabets L V and Frantseva A S 2017 Genetic algorithm in the problem of obtaining minimal Boolean functions representations in the class of Kronecker forms Soft Computing and Measurements (SCM), 2017 XX IEEE International Conf. 4034
[7] Vinokurov S F, Ryabets L V, Todikov S I and Frantseva A S 2017 Algorithm for constructing minimal representations of multiple-output Boolean functions in the reversible logic circuits Soft Computing and Measurements (SCM), 2017 XX IEEE International Conf. 5413
[8] Ilyin B, Kazimirov A, Panteleyev V, Reimerov S and Semicheva N 2017 On evolutionary algorithms for Boolean functions minimization *Soft Computing and Measurements (SCM)*, 2017 XX IEEE Int. Conf. 4002

[9] Kazimirov A and Reimerov S 2016 On Genetic Algorithms and Neural Networks for Boolean Functions Minimization *Proc. of the XIX Int. Conf. on Soft Computing and Measurement, SCM* 2601

[10] Kazimirov A S 2005 Bounding of LP-equivalences classes number of Boolean functions *Vestnik Buryatskogo universiteta: Matematika i informatika. Ulan-Ude, Buryatskiy gosudarstvenny universitet* 13 1722

[11] Koda N 1993 LP equivalence class of logic functions *IFIP 10.5 Workshop on Application of the Reed-Muller expansion in Circuit Design* 99-106

[12] Vinokurov S F and Kazimirov A S 2009 Operators classes enumeration of Boolean functions *The Bulletin of Irkutsk State University. Series Mathematics* 2 4055

[13] Kazimirov A S and Reimerov S U 2010 Computation complexity bounding of polynomial representation of Boolean functions *The Bulletin of Irkutsk State University. Series Mathematics* 4 33-43