Logic Minimization using Exclusive OR Gates

Valentina Ciriani
Dipartimento di Informatica
Corso Italia, 40 Pisa, Italy
ciriani@di.unipi.it

ABSTRACT

Recently introduced pseudoproducts and Sum of Pseudoproduct (SPP) forms have made possible to represent Boolean functions with much shorter expressions than standard Sum of Products (SP) forms [5]. A pseudo product is a product (AND) of Exclusive OR (EXOR) factors, and an SPP form is a sum (OR) of pseudoproducts. The synthesis of SPP minimal forms requires greater effort than SP minimization. In this paper we present a new data structure for this problem, leading to an efficient minimization method for SPP forms implemented with an exact algorithm and an heuristic. Experimental results on a classical set of benchmarks show that the new algorithms are fast, and can be applied to "complex" functions with a reasonable running time.

1. INTRODUCTION

A crucial task in logic synthesis is to produce efficient implementations of single or multi-output Boolean functions. In particular, Sum of Product (SP or two-level logic) minimization has been one of the most studied problem in computer science (see for example [2, 7, 9]).

Three-level logic is a good trade-off between the speed of two-level logic and the compactness of multi-level logic. Tree-level logic minimization is an harder problem than two-level logic minimization. In fact, the difficulty sharply increases with the number of levels. On the other hand, the number of products in a three-level network can be significantly smaller then the number of products of a two-level network. A relevant three-level minimization algorithm was given by Malik, Harrison and Brayton in [6]. They studied three-level network of the form $f = g_1 \circ g_2$ where g_1 and g_2 are SP forms and $\circ$ denotes a *binary* operation. Some heuristics in the case of $\circ = XOR$ (called *AND-OR-EXOR networks*) have been studied, for example, in [3, 4].

In a recent paper [5] Luccio and Pagli introduced a new three level form (*Sum of Pseudoproducts* or *SPP*) to express Boolean functions using Exclusive OR gates (EXOR), as a direct generalization of SP expressions. A *pseudoproduct* is a product (AND) of Exclusive OR (EXOR) factors, and an *SPP form* is a sum (OR) of pseudoproducts. For example, $(x_0 \oplus \overline{x}_1) \cdot x_4 \cdot (x_0 \oplus x_3 \oplus \overline{x}_6) + x_4 \cdot \overline{x}_3 + (x_0 \oplus x_2 \oplus x_3) \cdot (x_2 \oplus x_4 \oplus x_5)$ is an SPP form. An arbitrary Boolean function can be expressed as a disjunction of pseudoproducts, giving rise to a sum of pseudoproducts form. Experimental results show that the average size of SPP forms is approximately half the size of the corresponding SP forms. In the worst case, SP and SPP forms coincide. For example in table 1 the number of literals in the minimal SP expression of the benchmark function *adr4* is 4,72 times the number of literals in the minimal SPP form, while the function *newtpla2* yields identical results in SPP and SP minimization. On the other hand, minimization of SPP forms requires a greater effort than SP optimization. In this paper, we present the first algorithms ad hoc for SPP minimization.

SPP and SP forms have similar definitions and properties which we review here. A product P is an *implicant* of the Boolean function F iff $P \subseteq F$ (Boolean functions are treated as sets of points of B^n). An *EXOR factor* is a single variable, or a string of variables connected by EXOR's. A product P of EXOR factors is a *pseudoproduct* of the Boolean function F iff $P \subseteq F$. An implicant (pseudoproduct) P of F is *prime* iff no other implicant (pseudoproduct) P' of F exists such that $P \subseteq P'$.

Let X and Y be two sets, R be a relation on $X \times Y$, and $C : Y \to \mathbb{N}$ be a cost function over Y. We say that $Y' \subseteq Y$ *covers* X iff, for all $x \in X$, there exists $y \in Y'$ such that xRy holds. The *set covering problem* $\langle X, Y, R \rangle$ consists in finding a subset Z of Y such that Z covers X and $\sum_{z \in Z} C(z)$ is minimum. Minimal SP (SPP) covers of F are made of prime implicants (pseudoproduct), and SP (SPP) *minimization* is a set covering problem on the set of prime implicants (pseudoproduct) with cost function given by the number of factors or literals.

A Quine-McCluskey-like procedure is indicated in [5] as a possible algorithm for the synthesis of SPP forms. This algorithm fails in practice in minimizing functions (from the Espresso benchmark suite [10]) with a too large number of prime pseudoproducts (e.g., more than 15.000). In this paper, we present a new exact minimization algorithm and an incremental heuristic ad hoc for SPP forms to synthesize a much larger collection of Boolean functions. We introduce the *structure* of a pseudoproduct P as its algebraic expression without complementations, and prove that the union of two pseudoproducts with the same structure is still a pseudoproduct. This property is then used to design our

DAC 2001 June 18-22, 2001, Las Vegas, Nevada, USA.

minimization algorithms, which are based on a new data structure called *partition trie*. As in the Quine-McCluskey approach, the generation of prime pseudoproducts is performed in steps by successive unions of pseudoproducts, but here only the ones with the same structure are compared. For this purpose the pseudoproducts found in each step are grouped in sets, each of which contains pseudoproducts with the same structure, to reduce the computational work drastically. The previous algorithm of [5] performs approximately $|X^i|^2/2$ comparisons at step i, where X^i is the set of pseudoproducts generated at step $i-1$. In our method X^i is partitioned into disjoint subsets $X^i_1, X^i_2, \ldots, X^i_k$ where $1 \leq k \leq |X^i|$, hence the total number of comparisons is approximately $(|X^i_1|^2 + |X^i_2|^2 + \ldots + |X^i_k|^2)/2$, and this number is in general much smaller than $|X^i|^2/2$ as proved in our experiments. Indeed we perform a minimum number of comparisons, in the sense that each couple of pseudoproducts considered is unified to generate a new pseudoproduct for the next step. This increase of speed also allows us to deal with a very high number of prime pseudoproducts (up to 300.000 in practice).

In the next section, we recall the basic notions of pseudocubes and canonical expression, and some properties of pseudoproducts. In section 3 we introduce the concept of structure, we describe the partition trie data structure and the new minimization algorithms. Section 4 gives some experimental results demonstrating the effectiveness of the new method.

2. PRELIMINARIES

Pseudocubes, pseudoproducts and SPP forms studied in [5] are generalizations of cubes, products and SP forms. The definitions and properties in this section are taken from that paper.

Let $\boldsymbol{u}$ be a Boolean vector (in fact all our vectors are Boolean). Vector $\overline{\boldsymbol{u}}$ denotes the elementwise complementation of $\boldsymbol{u}$. Vector $\widehat{\boldsymbol{u}}$ denotes $\boldsymbol{u}$ or $\overline{\boldsymbol{u}}$. Vectors $\mathbf{0}$ and $\mathbf{1}$ are made up of all 0's or all 1's, respectively, and are called *constant* vectors. Vector $\boldsymbol{uv}$ is the concatenation of $\boldsymbol{u}$ and $\boldsymbol{v}$. A vector $\boldsymbol{u}$ of 2^m elements, $m \geq 0$, is *normal* if $m = 0$, or $m > 0$ and $\boldsymbol{u} = \boldsymbol{v}\widehat{\boldsymbol{v}}$ with $\boldsymbol{v}$ (hence $\widehat{\boldsymbol{v}}$) normal. For example all the columns in the matrix of figure 1 are normal vectors.

In the following, a matrix will always have 2^m rows and n columns, with $n \geq 1$ and $0 \leq m \leq n$. A matrix M is *normal* if has distinct rows, and all the columns are normal. A normal matrix is *canonical* if its rows, interpreted as binary numbers, are arranged in increasing order (the matrix of figure 1 is canonical). A normal vector $\boldsymbol{u} = \boldsymbol{v_0} \ldots \boldsymbol{v_{2^{m-k}-1}}$ is *k-canonical*, $0 \leq k < m$, if $\boldsymbol{v_i} = \mathbf{0}$ for i even, and $\boldsymbol{v_i} = \mathbf{1}$ for i odd (note that $|\boldsymbol{v_i}| = |\boldsymbol{v_j}|$ for all i, j). In figure 1, c_0 is 2-canonical, c_2 is 1-canonical, and c_4 is 0-canonical. A canonical matrix M contains m columns $c_{i_0}, \ldots c_{i_{m-1}}$ of increasing indices, such that c_{i_j} is $(m-j-1)$-canonical for $0 \leq j \leq m-1$. In a canonical matrix M the columns $c_{i_0}, \ldots c_{i_{m-1}}$ are called *canonical columns* of M. The other columns are the *non-canonical* ones (see figure 1).

A set of m points of the Boolean space B^n can be arranged in a $m \times n$ matrix whose rows correspond to the points and the columns correspond to the variables describing B^n. Canonical and non-canonical columns correspond to *canonical* and *non-canonical variables*, respectively. A *pseudocube* of degree m is a set of 2^m points whose matrix is canonical up to a row permutation. The matrix of figure 1 represents a

	$\mathbf{c_0}$	$\mathbf{c_1}$	$\mathbf{c_2}$	$\mathbf{c_3}$	$\mathbf{c_4}$	$\mathbf{c_5}$
$\mathbf{r_0}$	0	1	0	1	0	1
$\mathbf{r_1}$	0	1	0	1	1	0
$\mathbf{r_2}$	0	1	1	0	0	1
$\mathbf{r_3}$	0	1	1	0	1	0
$\mathbf{r_4}$	1	1	0	0	0	0
$\mathbf{r_5}$	1	1	0	0	1	1
$\mathbf{r_6}$	1	1	1	1	0	0
$\mathbf{r_7}$	1	1	1	1	1	1

Figure 1: A canonical matrix with 2^3 rows and $n = 6$ columns. The canonical columns are c_0, c_2, and c_4. The matrix represents a pseudoproduct in B^6.

pseudocube of eight points in B^6. The points represented in figure 1 are: $\overline{x}_0 x_1 \overline{x}_2 x_3 \overline{x}_4 x_5$, $\overline{x}_0 x_1 \overline{x}_2 x_3 x_4 \overline{x}_5$, $\overline{x}_0 x_1 x_2 \overline{x}_3 \overline{x}_4 x_5$, $\overline{x}_0 x_1 x_2 \overline{x}_3 x_4 \overline{x}_5$, $x_0 x_1 \overline{x}_2 \overline{x}_3 \overline{x}_4 \overline{x}_5$, $x_0 x_1 \overline{x}_2 \overline{x}_3 x_4 x_5$, $x_0 x_1 x_2 x_3 \overline{x}_4 \overline{x}_5$, $x_0 x_1 x_2 x_3 x_4 x_5$. Note that a cube is a special case of pseudocube where the non-canonical columns are constant. For a point $s \in B^n$ and a subset of variables α, the *transformed point* $\alpha(s)$ is obtained from s by complementing the variables in α. For a pseudocube (or, in general, a set of points) P, the *transformed set* $\alpha(P)$ is the set $\{\alpha(s) : s \in P\}$.

PROPOSITION 1. *Let P_1 be a pseudocube of degree m, and α be a subset of non-canonical variables. Then $P_2 = \alpha(P_1)$ is a pseudocube of degree m, with $P_1 \cap P_2 = \emptyset$, and $P_1 \cup P_2$ is a pseudocube of degree $m+1$. Moreover, $P_1 \cup P_2$ is a pseudocube only if $P_2 = \alpha(P_1)$ for some α.*

Let M_1, M_2 be the canonical matrices of P_1, P_2, with $P_2 = \alpha(P_1)$. Matrices M_1 and M_2 have the same canonical variables $x_{p_0}, \ldots, x_{p_{m-1}}$, and non-canonical variables $x_{p_m}, \ldots, x_{p_{n-1}}$. The canonical matrix M of $P = P_1 \cup P_2$ has the same canonical variables of M_1, M_2, plus the variable $x_k \in \alpha$ of minimum index.

The characteristic function of a pseudocube is called *pseudoproduct*, and has the concise algebraic expression introduced in the following definition:

DEFINITION 1. *Let P be a pseudocube of degree m in B^n; let M be the canonical matrix of P; and let $x_{p_0}, \ldots, x_{p_{m-1}}$ and $x_{p_m}, \ldots, x_{p_{n-1}}$ be the canonical and non-canonical variables respectively, with these two sets ordered for increasing values of the indices. The* canonical expression CEX(P) *associated with P is given by $f_0 \cdot f_1 \cdot \ldots \cdot f_{n-m-1}$, where each f_i, $0 \leq i \leq n-m-1$, is an EXOR factor containing the following variables:*

1. *the canonical variables x_{p_j}, $0 \leq j \leq m-1$, such that $M[0, p_{m+i}] \neq M[2^{m-j-1}, p_{m+i}]$; these variables are ordered for increasing indices;*
2. *the non-canonical variable $x_{p_{m+i}}$, if $M[0, p_{m+i}] = 1$; or $\overline{x}_{p_{m+i}}$, if $M[0, p_{m+i}] = 0$.*

In figure 1 column c_i corresponds to variable x_i $(0 \leq i \leq 5)$. The canonical expression for the pseudocube is

$$CEX = x_1 \cdot (x_0 \oplus x_2 \oplus x_3) \cdot (x_0 \oplus x_4 \oplus x_5)$$

Since the non-canonical variables are x_1, x_3, x_5, the CEX expression is given by the product of three EXOR factors: f_0 containing $\widehat{x}_1$, f_1 containing $\widehat{x}_3$, and f_2 containing $\widehat{x}_5$. Any EXOR factor can also contain some canonical variables as described in rule 1 of definition 1. For example, since the column c_1 is constant, f_0 does not contain any canonical variable. Whereas f_1 contains x_0 and x_2, and f_2 contains

x_0 and x_4. Let's associate row r_1 with the canonical variable x_4, row r_2 with the canonical variable x_2, and row r_4 with the canonical variable x_0. Following rule 1 of definition 1, f_1 contains x_0 and x_2 but not x_4 because in the column c_3 the first row $r_0 = 1$ is different by row $r_2 = 0$ (corresponding to x_2) and by row $r_4 = 0$ (corresponding to x_0), but is equal to row $r_1 = 1$ (corresponding to x_4).

Note that $CEX(P)$ depends on the arbitrary order of the variables, or on the values of the variables indices. That is a pseudocube have in general many different pseudoproduct expressions, but only one of them is canonical once the ordering of the variables is fixed.

3. MINIMIZATION ALGORITHMS

We now introduce some more properties of pseudocubes used in the minimization algorithms, and describe the partition trie data structure, which is the basis of the minimizations procedures. Finally, we give the exact algorithm and the heuristic.

3.1 The structure of a pseudocube

DEFINITION 2. *Let P be a pseudocube. The* structure *of P, denoted by* STR(P), *is the CEX expression of P without complementations.*

For example, if $CEX(P) = (x_0 \oplus x_1 \oplus \overline{x}_3) \cdot (x_0 \oplus x_4 \oplus x_5) \cdot \overline{x}_7$ we have $STR(P) = (x_0 \oplus x_1 \oplus x_3) \cdot (x_0 \oplus x_4 \oplus x_5) \cdot x_7$. Note that two distinct pseudocubes with the same structure are disjoint.

A well known Boolean property states that the union (or sum) of two products is a product if and only if they are composed by the same literals and differ by exactly one complementation. This also implies that the corresponding subcubes are disjoint. For example $x_0x_2\overline{x}_3x_6 + x_0\overline{x}_2\overline{x}_3x_6 = x_0\overline{x}_3x_6$. The corresponding property of pseudocubes is the following:

THEOREM 1. *The union of two pseudocubes P_1, P_2 is a pseudocube if and only if $STR(P_1) = STR(P_2)$.*

PROOF. *If part.* Let α be the (possibly empty) set of non-canonical variables that have different complementation in $CEX(P_1)$ and $CEX(P_2)$. From definition of α follows that $P_2 = \alpha(P_1)$ (or $P_1 = \alpha(P_2)$). By proposition 1, we have that $P_1 \cup P_2$ is a pseudocube. *Only if part.* By proposition 1, P_2 is equal to $\alpha(P_1)$. By definition of α, the pseudocube $\alpha(P_1)$ has the same structure of P_1. □

For example, let

$$(x_0 \oplus \overline{x}_1) \cdot x_4 \cdot (x_0 \oplus x_2 \oplus \overline{x}_5) \cdot (x_3 \oplus x_6) \cdot (x_3 \oplus x_8) \quad (1)$$

$$(x_0 \oplus x_1) \cdot \overline{x}_4 \cdot (x_0 \oplus x_2 \oplus x_5) \cdot (x_3 \oplus x_6) \cdot (x_3 \oplus \overline{x}_8) \quad (2)$$

be the CEX expressions of two pseudocubes (P_1 and P_2) in B^9 with canonical variables x_0, x_2, x_3, x_7, and $\alpha = \{x_1, x_4, x_5, x_8\}$ (recall that α is the subset of non-canonical variables that contains only variables with different complementation in the two CEX expressions). Note that the two CEX expression have the same structure:

$$(x_0 \oplus x_1) \cdot x_4 \cdot (x_0 \oplus x_2 \oplus x_5) \cdot (x_3 \oplus x_6) \cdot (x_3 \oplus x_8)$$

and $P = P_1 \cup P_2$ is a pseudocube by theorem 1.

We now rephrase the union algorithm of [5] using the notion of structure. Let $x_{i_0}, \ldots x_{i_h}$ be the non-canonical variables of P_1 and P_2 in increasing order of their indices. Therefore $CEX(P_1) = \prod_{j=0}^{h} f_{i_j}^1$ and $CEX(P_2) = \prod_{j=0}^{h} f_{i_j}^2$, where $f_{i_k}^1$ and $f_{i_k}^2$ are the EXOR factors containing the non-canonical variable x_{i_k}. We then derive the CEX expression for $P = P_1 \cup P_2$ by the following algorithm:

ALGORITHM 1. ***Union*** *(build $CEX(P)$ from $CEX(P_1)$ and $CEX(P_2)$)*

if $STR(P_1) = STR(P_2)$ **then**
  let x_{i_k} be the variable of smaller index in α
  for $j = 0$ **to** h **do**
    if ($x_{i_j} \in \alpha$ **and** $i_j \neq i_k$)
      then $f_{i_j}^p := NORM\_EXOR(f_{i_j}^2, f_{i_k}^1)$
    if ($x_{i_j} \notin \alpha$) **then** $f_{i_j}^p := f_{i_j}^2$
  $CEX(P) := \prod_{\substack{j=0 \\ j \neq k}}^{h} f_{i_j}^p$

The expression $NORM\_EXOR(f_1, f_2)$ is the *normalized EXOR expression* between the EXOR factors f_1 and f_2. For example, let $f_1 = (x_0 \oplus x_2 \oplus x_5)$ and $f_2 = (x_0 \oplus \overline{x}_1)$ be two EXOR factors. The expression $f_1 \oplus f_2 = x_0 \oplus \overline{x}_1 \oplus x_0 \oplus x_2 \oplus x_5$ can be reduced (normalized), using some properties of EXOR[1], to $f_1 \oplus f_2 = x_1 \oplus x_2 \oplus \overline{x}_5$, which is $NORM\_EXOR(f_1, f_2)$.

As an example of application of the union algorithm, consider P_1 and P_2 with expressions (1) and (2). Since $\alpha = \{x_1, x_4, x_5, x_8\}$, we have $x_{i_k} = x_1$ and the canonical variables of $CEX(P)$ are x_0, x_1, x_2, x_3, x_7. Then the canonical expression is

$$(x_0 \oplus x_1 \oplus x_4) \cdot (x_1 \oplus x_2 \oplus \overline{x}_5) \cdot (x_3 \oplus x_6) \cdot (x_0 \oplus x_1 \oplus x_3 \oplus x_8).$$

Note that the factor $(x_0 \oplus x_1)$ does not appear in $CEX(P)$ (the variable x_1 is canonical in P). Any other factor f of $CEX(P_2)$ that has different complementation in $CEX(P_1)$ and $CEX(P_2)$ appears in $CEX(P)$ as $NORM\_EXOR(f, (x_0 \oplus \overline{x}_1))$ (see for example $\overline{x}_4$, which becomes $(x_0 \oplus x_1 \oplus x_4)$). Any factor that has equal complementation in $CEX(P_1)$ and $CEX(P_2)$ appears unchanged in $CEX(P)$ (see for example $(x_3 \oplus x_6)$).

The time required by algorithm Union is linear in the size of the input expressions.

3.2 Partition tries

Partition tries are the basis of two new minimization algorithms to be discussed in sections 3.3 and 3.4. A partition trie is a labeled rooted tree used to represent both the structures and the CEX expressions of a set of pseudoproducts.

An internal node (except for the root) of the partition trie can be either a *C-node* (canonical node) or an *NC-node* (non-canonical node), and is labeled with the variable x_i. The leaves are Boolean vectors. The root has no label.

In a partition trie, any path from root to leaf represents the structure of a CEX expression. The vector L in the leaf of the path represents the complementations of the non-canonical variables in the NC-nodes of the path. That is, if $L[i] = 0$ then the i-th (in increasing order) non-canonical variable in the CEX expression is complemented; if $L[i] = 1$ such a variable is not complemented.

Let f_i be the i-th EXOR factor of a CEX expression C in a partition trie P. In the path representing C, the EXOR factors are in increasing order of the non-canonical variables. In every f_i, the canonical variables are in increasing order, and

[1]Namely, $x \oplus x = 0$, $0 \oplus x = x$, $\overline{x} \oplus y = x \oplus \overline{y} = \overline{x \oplus y}$

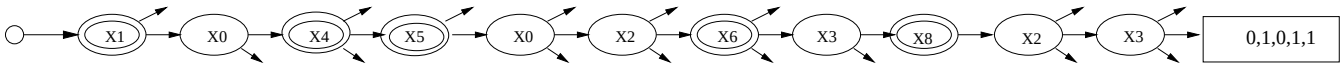

Figure 2: A path in a partition tree representing the CEX expression: $(x_0 \oplus \overline{x}_1) \cdot x_4 \cdot (x_0 \oplus x_2 \oplus \overline{x}_5) \cdot (x_3 \oplus x_6) \cdot (x_2 \oplus x_3 \oplus x_8)$. C-nodes and NC-nodes are circled and double-circled, respectively.

the first node is an NC-node representing the non-canonical variable x_k; all the others nodes, in increasing order, are the C-nodes of the canonical variables in f_i.

For example the path in figure 2 represents a CEX expression. Unlike for the corresponding CEX expression, any EXOR factor in a path of a partition trie starts with its non-canonical variable. We have:

PROPERTY 1. *Any two leaves with the same parent represent CEX expressions with the same structure.*

We call this kind of trees partition tries because a set of CEX expressions is partitioned into subsets containing expressions with the same structure; and because they are particular tries with two different kinds of nodes (see [8] for details on tries).

The children of each node are ordered as follows: first the NC-nodes in increasing order of their labels, second the C-nodes in increasing order of their labels and, finally, the leaves. The insertion of a variable x_i as a child of a given node N is an extension of the insertion procedure for a trie, taking into account the constraints given by the labels.

The insertion of an EXOR factor X as a child of a given node N is as follows: first, insert the non-canonical variable as a child of N; second, insert the first (in increasing order) canonical variable as a child of such a non-canonical variable; finally, insert each of the other canonical variables, in increasing order, as a child of the last canonical variable inserted. Insertion of a CEX expression C corresponds to the insertion of its EXOR factors in the increasing order of the non-canonical variables.

The search for a CEX expression C in the partition trie is done by examining the tree from the root.

Note that any internal node in a partition trie can have at most $2n$ internal nodes as children, since any such a child is a C-node or an NC-node, representing a variable, or a leaf. Partition tries will be used to represent pseudoproducts in the minimization procedure.

3.3 The exact minimization method

We focus our attention to the synthesis of SPP forms with minimal number of literals. As usual pseudoproducts will be built by pairwise composition of pseudoproducts of smaller degree, but we memorize pseudoproducts in a much more efficient way than before. As usual the number of literals in a CEX expression may depend on the ordering of the variables, therefore minimization holds relatively to a chosen order. Unlike for products, a pseudoproduct built by composition of two pseudoproducts of smaller degree dose not always contains less literals than the two components. For example the CEX expression of the union of (1) and (2) is

$$(x_0 \oplus x_1 \oplus x_4) \cdot (x_1 \oplus x_2 \oplus \overline{x}_5) \cdot (x_3 \oplus x_6) \cdot (x_0 \oplus x_1 \oplus x_3 \oplus x_8)$$

which contains 12 literals, while (1) and (2) have 10 literals each. As a consequence, the construction of an SPP form with minimal number of literals cannot be limited to the set of prime pseudoproducts, but to a larger one. We pose:

DEFINITION 3. *The set of* extended prime pseudoproducts (EPPP) *is the union of the set of prime pseudoproducts and the set of pseudoproducts which are not covered by any other pseudoproduct whose expression contains less literals.*

Both the minimization algorithm proposed in [5], and the new algorithm presented here, are composed of two main steps: first, we build the EPPP set; second, we solve a set covering problem where the rows X represent the points, the columns Y represent the EPPP set, and the cost of a column is the number of literals in the associated CEX expression.

The critical novelty of our algorithm is the data structures used in the construction of the EPPP set, now based on partition tries. In fact, we do not need to compare pairwise all the CEX expressions of degree k, due to theorem 1. In step k we use a partition trie to memorize all the CEX expressions generated, where all and only the leaves with common parent have the same structure and can be unified. This property is implemented as follows:

ALGORITHM 2. ***Exact minimization***

1. insert the CEX expressions of all the pseudoproducts of degree $k = 0$ (single points) in a partition trie P_0;
2. for increasing k, generate a *new* partition trie P_{k+1} from the union of all the leaves that have common parent in P_k; discard from P_k a pseudoproduct if its CEX expression contains h literals, and is combined into one of degree $k+1$ whose expression contains $\leq h$ literals;
3. make a selection with minimal number of literals from among the pseudoproducts retained in step 2 (i.e. solve the set covering problem).

Union algorithm 1 is used in step 2, with the automatic satisfaction of the initial test for structure of leaves with the same parent. Algorithm 2 does not decrease the worst-case complexity of the minimization procedures, since the problem is NP hard for SP and remains already NP hard for SPP. Comparing the method of [5] with algorithm 2, however, we find that the second is more efficient in practice because inserting a CEX expression in the partition trie and unifying two leaves with common parent involves a much smaller number of CEX expressions. In fact the original algorithm performs $|X^i|(|X^i|-1)/2$ comparisons in step i, where X^i is the set of pseudoproducts generated in step $i-1$. Algorithm 2, instead, partitions X^i into disjoint sets $X^i_1, X^i_2, \ldots, X^i_k$, where $1 \leq k \leq |X^i|$ and $X^i = \bigcup_{1 \leq j \leq k} X^i_j$. Since the pairwise comparisons are performed only inside each set, the total number of comparison is $|X^i_1|(|X^i_1|-1)/2 + |X^i_2|(|X^i_2|-1)/2 + \ldots + |X^i_k|(|X^i_k|-1)/2$, which can be much smaller than $|X^i|(|X^i|-1)/2$ for $k > 1$. The new algorithm, in practice, does not perform any comparison, because every couple of pseudoproducts considered will be unified and inserted in the partition trie of the next step. Experimental evidence of the good performance of algorithm 2 is given in section 4.

function	SP			SPP		
	#PI	#L	#P	#EPPP	#L	#PP
addm4	352	1299	212	191133	520	74
adr4	75	340	75	7158	72	14
dist	279	829	150	48753	422	64
ex5	650	828	307	273695	723	253
exps	950	3007	499	63083	1918	273
life	224	672	84	2100	144	18
lin.rom	827	2165	451	39280	1235	227
m3	212	693	131	13768	423	74
m4	441	984	211	110198	646	123
max128	338	795	191	15504	492	108
max512	416	923	154	298623	517	76
mlp4	206	709	143	24982	318	61
newcond	55	208	31	46889	122	15
newtpla2	15	74	15	17146	74	15
p1	205	362	100	476360	232	44
prom2	2298	6647	940	341557	3477	383
radd	75	340	75	6600	72	14
root	133	346	71	37324	220	39
test1	1066	1000	184	444407	534	73

Table 1: Experimental results. #PI and #EPPP are the total numbers of prime implicants and EPPP's. #L is the number of literals in the minimal expression. #P and #PP are the numbers of prime implicants and EPPP's in the minimal expression.

3.4 The heuristic

The second step of algorithm 2 is a bottleneck, because it requires the computation of all the prime pseudoproducts. Alternatively we can use an incremental heuristic that, at any step, computes a subset of non decreasing size of pseudoproducts, and a more and more accurate upper bound of the solution (the last step gives the SPP minimal form). Unlike in algorithm 2, the input is an arbitrary cover of the given function F: below we use the set of prime implicants of the SP minimization of F, as this set is much faster to obtain than the set of prime pseudoproducts.

We now give a characterization of the sub-pseudocubes of a pseudocube (for a proof see [1]):

THEOREM 2. *Let $A_1 A_2 \dots A_q$ be a CEX expression of a pseudocube R, of degree m, and let $x_1, x_2, \dots, x_m$ the canonical variables of R. The set of expression $A_1 A_2 \dots A_q A_{q+1}$, with $A_{q+1} = y_1 \oplus y_2 \oplus \dots \oplus \widehat{y_k}$ and $\forall i \in [1, \dots, k] : y_i \in \{x_1, x_2, \dots, x_m\}$, represents all the distinct pseudoproducts P of degree $m-1$ with $P \subset R$ (the cardinality of this set is $2^{m+1} - 2$)*[2].

The heuristic consists of four main phases:

ALGORITHM 3. ***Heuristic***

1. Initialize the partition tries with the prime implicants: there are n partition tries $(P_0, \dots, P_{n-1})$, where n is the number of variables in F, and any implicant with i factors (i.e., with degree $n-i+1$) is inserted in the corresponding partition trie P_{n-i}.
2. Perform a "descendant phase" of k steps $(0 \le k < n)$, where the parameter k is a measure of the computational work decided for the heuristic ($k = n-1$ means that we are looking for the optimal SPP solution). In step i $(1 \le i \le k)$, for all the pseudoproducts R in partition trie P_{n-i} (R has degree $n-i+1$), compute all the distinct pseudoproduct P of degree $n-i$ with $P \subset R$, according to theorem 2. Insert in P_{n-i-1} any P.
3. "Ascendant phase". Starting from P_0, compute $P_1, \dots, P_{n-1}$ according to the second step of algorithm 2.
4. Solve the set covering problem.

Note that, if $k = 0$ the descendant phase is not performed, but in the ascendant phase we can find enough pseudoproducts to obtain a significant upper bound of the SPP form. For example, letting $x_1 x_2 \overline{x}_4$ and $\overline{x}_1 x_2 x_4$ be members of the set of prime implicants, the ascendant phase computes $x_2(x_1 \oplus x_4)$, and this expression can be present in the SPP form instead of $x_1 x_2 \overline{x}_4$ and $\overline{x}_1 x_2 x_4$. Unlike the exact algorithm, the heuristic (with $k < n-1$) does not compute the entire set of prime pseudoproducts, and produces an SPP form indicated by SPP_k for the applied value of k. However, in practice is more efficient, as described below.

[2]Note that the expressions $A_1 A_2 \dots A_q A_{q+1}$ are not necessarily in CEX form, but can be easily transformed in the equivalent CEX expressions.

function	#L	Time alg. in [5]	Time alg. 2
cs8(1)	124	783	4
cs8(2)	93	12945	21
addm4(2)	101	74	2
addm4(4)	104	*	146
prom1(15)	213	40	1
prom1(31)	278	*	41
max128(20)	7	4097	7
m3(3)	13	7039	9
m4(0)	5	*	4023
risc(2)	12	10	1
ex5(50)	9	*	3973
max512(5)	208	*	204

Table 2: CPU times (in seconds) of the two minimization algorithms for the construction of the EPPP set. The results are relative to single outputs, that is cs8(1) corresponds to the first output of cs8. A star indicates that the original algorithm did not terminate after 2 days. #L is the number of literals in the minimal expression.

4. EXPERIMENTAL RESULTS

Our minimization method has been tested on a range of functions taken from the ESPRESSO benchmark suite [10]. Table 1 compares SP expressions and SPP forms obtained with algorithm 2. The different outputs of each function have been minimized separately. All the functions in table 1 have a huge number of EPPP's and could not be minimized with the algorithm of [5] (the program did not stop within 172800 seconds, or 2 days, on our Pentium III 450 machine). This proves the crucial role of partition tries. Since we used some heuristics in solving the set covering problem, the number of literals and factors in the expressions are upper bounds for the minimal solution. Note that the size of the minimal SPP form is, on the average, one half of the corresponding SP form. The functions *adr4*, *life* and *radd* show a greater improvement of SPP over SP, while the function *newtpla2* yields identical results in SPP and SP, although the number of EEEP's is huge.

Table 2 compares the CPU time for the construction of the EPPP set for the two exact minimization methods. CPU times are reported in seconds on a Pentium III 450 machine. The tests are taken from the ESPRESSO benchmark suite and

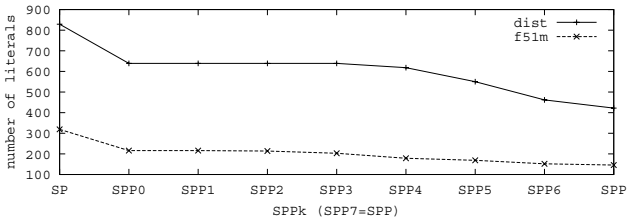

Figure 3: Number of literals in the SP form and in the SPP_k forms of the benchmark functions *dist* and $f51m$.

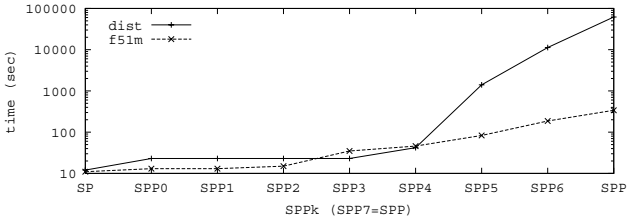

Figure 4: CPU time (in seconds) for the synthesis of the SP form and SPP_k forms of the benchmark functions *dist* and $f51m$ (logarithmic scale).

for some outputs of an 8-bit carry-save adder (*cs8(1)* and *cs8(2)*). These examples show the great increase of speed of algorithm 2 versus the original one.

Table 3 gives the results of the heuristic for $k = 0$. From this table we see that the number of literals of the SPP_0 forms is in average $(|SP| - |SPP|)/2$, where $|SP|$ (resp. $|SPP|$) is the number of literals in the minimal SP (resp. SPP) form (see the columns in bold of table 3). The computational time is drastically reduced. The number of literals and the CPU time of the SP and SPP_k forms of the functions *dist* and *f51m* are plotted in figure 3 and 4, respectively. Almost all the functions from the Espresso benchmark suite have analogous plots. Function *dist* in figures 3 and 4 has 829 literals in the SP form, and 639 in the SPP_0 form, and the CPU times of the two forms are 12 and 23 seconds, respectively. SPP_6 form has 462 literals, and the CPU time required for its synthesis is 11,285 seconds. Whereas the number of literals of the exact form ($SPP_7 = SPP$) is 422, this form is obtained in 61,925 seconds. It is significant to notice that, for values of k near to $n - 1$ (exact SPP form), the number of literals from SPP_k to SPP_{k+1} decreases slowly, but the CPU time for SPP_{k+1} synthesis is much greater than the one for SPP_k. Therefore we claim that SPP_k forms are reasonable upper bounds of the exact SPP forms for small values of k.

5. CONCLUSIONS

In this paper we have presented, for the first time, two specific procedures for the SPP minimization of Boolean functions based on partition tries. This method allows to deal with functions whose number of prime pseudoproducts is too high to be handled by previous algorithms. The number of prime pseudoproducts is in general much greater then the number of prime implicants and we are still not able to deal, in reasonable time, with the hardest functions in Espresso benchmarks. Therefore, our future investigations will be concentrated on algorithms whose complexity no longer depends on the number of pseudoproducts to manipulate. Furthermore, we plan to compare SPP forms with other three level forms.

		SPP_0		**SPP**	**(exact)**
function	Av	#L	Time	#L	Time
alu	*	**41**	51050	*	*
addm4	**910**	**939**	16	520	27340
add6	*	**1212**	7454	*	*
amd	*	**905**	96826	*	*
dist	**626**	**639**	23	422	61925
f51m	**233**	**216**	13	146	339
max512	**720**	**693**	40	517	12609
max1024	*	**1098**	192	*	*
mlp4	**586**	**643**	7	318	778
m4	**815**	**785**	64	646	18123
newcond	**165**	**166**	12	122	15587

Table 3: Experimental results for the heuristic with $k = 0$ vs. the exact algorithm. A star indicates that the exact algorithm did not terminate after 2 days. Av is $(|SP|-|SPP|)/2$, where $|SP|$ (resp. $|SPP|$) is the number of literals in the minimal SP (resp. SPP) form. #L is the number of literals in the forms.

6. ACKNOWLEDGMENT

The author would like to thank Fabrizio Luccio, Linda Pagli and Paolo Ferragina for many fruitful discussions, and Roberto Grossi for the precious comments on the drafts of this paper.

7. REFERENCES

[1] V. Ciriani. The characterization of the sub-pseudocubes of a pseudocube. Technical Report TR-00-02, Dipartimento di Informatica, Università di Pisa, 2000.

[2] O. Coudert. Doing Two-Level Logic Minimization 100 Times Faster. In *SODA*, pages 112–121, 1995.

[3] D. Debnath and T. Sasao. A Heuristic Algorithm to Design AND-OR-EXOR Three-Llevel Networks. In *Asia and South Pacific Design Automation Conference*, pages 69–74, 1998.

[4] E. Dubrova, D. Miller, and J. Muzio. AOXMIN-MV: A Heuristic Algorithm for AND-OR-XOR Minimization. In *4th Int. Workshop on the Applications of the Reed Muller Expansion in circuit Design*, pages 37–54, 1999.

[5] F. Luccio and L. Pagli. On a New Boolean Function with Applications. *IEEE Transactions on Computers*, 48(3):296–310, 1999.

[6] A. Malik, D. Harrison, and R. Brayton. Three-Level Decomposition with Application to PLDs. In *IEEE ICCD*, pages 628–633, 1991.

[7] P. McGeer, J. Sanghavi, R. Brayton, and A. Sangiovanni-Vincentelli. Espresso-Signature: A New Exact Minimizer for Logic Functions. *IEEE Transactions on VLSI*, 1(4):432–440, 1993.

[8] R. Sedgewick and P. Flajolet. *An introduction to the analysis of algorithms.* Addison-Wesley, 1996.

[9] T. Villa, T. Kam, R. Brayton, and A. Sangiovanni-Vincentelli. *Synthesis of finite state machines: logic optimization.* Kluwer Academic Publishers, 1997.

[10] S. Yang. Synthesis on Optimization benchmarks. User guide, Microelectronic Center of North Carolina, 1991. Benchmarks available at ftp://ftp.sunsite.org.uk/computing/general/espresso.tar.Z.