UNIVERSITÀ DI PISA
DIPARTIMENTO DI INFORMATICA

TECHNICAL REPORT: TR-10-21

Compact DSOP Forms Derived from SOP Expressions

Anna Bernasconi
Dept. of Computer Science
Università di Pisa, Italy
annab@di.unipi.it

Valentina Ciriani
Dept. of Information Technology
Università degli Studi di Milano, Italy
valentina.ciriani@unimi.it

Fabrizio Luccio
Dept. of Computer Science
Università di Pisa, Italy
luccio@di.unipi.it

Linda Pagli
Dept. of Computer Science
Università di Pisa, Italy
pagli@di.unipi.it

December 1, 2010

ADDRESS: via F. Buonarroti 2, 56127 Pisa, Italy. TEL: +39 050 2212700 FAX: +39 050 2212726

Compact DSOP Forms Derived from SOP Expressions

Anna Bernasconi
Dept. of Computer Science
Università di Pisa, Italy
annab@di.unipi.it

Valentina Ciriani
Dept. of Information Technology
Università degli Studi di Milano, Italy
valentina.ciriani@unimi.it

Fabrizio Luccio
Dept. of Computer Science
Università di Pisa, Italy
luccio@di.unipi.it

Linda Pagli
Dept. of Computer Science
Università di Pisa, Italy
pagli@di.unipi.it

***Abstract*—We give a new heuristic for Disjoint Sum-of-Products (DSOP) minimization of a Boolean function f, based on a new criterion for product selection. Starting from a Sum-of-Products (SOP) S of f, i.e., a set of cubes covering the minterms of f, we assign a weight $w(p)$ to each product (cube) p in S, where $w(p)$ depends on the intersection of p with the other cubes of S. We assign higher weights to the cubes that, if selected in a DSOP, would generate a higher fragmentation of the other cubes. Disjoint cubes are then selected for increasing value of w and decreasing size, recomputing the SOP on the residual function at different possible stages as a trade-off between quality of result and computational time. We also propose a heuristic for computing partial DSOP forms, i.e., SOP forms whose cubes cover exactly once a given subset of minterms of the function, and more than once the remaining minterms. A set of experiments conducted on major benchmark functions show that our method, with all its variants, always generates better results than the ones of previous heuristics, including the one based on a BDD representation of f. These experiments have also outlined how re-synthesizing the residual function in SOP form seems to be crucial for obtaining compact DSOPs.**

Keywords: Sum-of-Products, Disjoint Sum-of-Products, Cube selection, Cube fragmentation, Minimal form, ESPRESSO.

I. Introduction

Given a Boolean function f on n variables $x_1, x_2, ..., x_n$ in $\mathcal{B}^n$, a *Disjoint Sum-of-Products (DSOP)* of f is a set of products (ANDs) of subsets of literals whose sum (OR) equals f, such that no two products cover the same minterm of f. As each product is the mathematical expression for a cube in $\mathcal{B}^n$, a DSOP also represents a set of non intersecting cubes occupying the points of $\mathcal{B}^n$ in which $f = 1$. In fact we shall indifferently refer to products or cubes, and apply algebraic or set operations to them. We are interested in minimizing |DSOP|, i.e., finding a DSOP with a minimal number of products.

Besides its theoretical interest, DSOP minimization is relevant in the area of digital circuits for determining various properties of Boolean functions and for the synthesis of asynchronous circuits, as discussed for example in [2], [9], [10], [14]. DSOPs are indeed used as a starting point for the synthesis of *Exclusive-Or-Sum-Of-Products (EPSOP)* forms, and for calculating the spectra of Boolean functions.

For speeding an otherwise exceedingly cumbersome process an absolute minimum in general is not sought for, rather heuristic strategies for cube selection have been proposed, working on explicit product expressions [3], [13], or on a BDD representation of f [4]. Here we study a class of heuristic algorithms for DSOP minimization based on the new concept of "cube weight", and show that our results compare favorably with the ones of the other known heuristics. The starting set of cubes is the one of a SOP, found with standard heuristics. The SOP cubes may be eventually fragmented into non overlapping sub-cubes, giving rise to a largely unpredictable DSOP solution. The process may exhibit an exponential blow up in the number of fragments even dealing with theoretically minimal solutions, as for a function presented in [11] where $|\text{SOP}| = n/2$ and $|\text{DSOP}| = 2^{n/2} - 1$.

Another new characteristic of our heuristic is the idea of recomputing a SOP on the residual function at different possible stages of the disjoint minimization process, as a trade-off between quality of the result and computational time. We have observed experimentally that this strategy is crucial for obtaining compact DSOP forms.

Some applications may require to cover a subset of minterms of a Boolean function exactly once, while other minterms can be covered more than once. For example this is the case where the points in the on set of a function must be covered exactly once, while the points in the don't care set can be covered any number of times [9], [5]. We show how our heuristic can be modified in order to efficiently compute such *partial DSOP forms*.

A preliminary version of this paper has been presented in *IWSPB* [1]. In the present journal version we extend the study to different heuristic algorithms, discuss the synthesis of incompletely specified functions, and generalize the approach to the Partial DSOP synthesis. The experimental results have been extended accordingly. The paper is organized as follows. In the following Section II we define the weight of a product p as a functions of the number of fragments possibly induced

on other cubes by the selection of p. In Section III we show how this weight can be exploited for building a class of minimization heuristics. Section IV presents a strategy for partial DSOP synthesis. In Section V we present and discuss the computational results obtained by applying the proposed heuristic to the standard ESPRESSO benchmark suite [16], and comparing these results with other published data. The paper is concluded in Section VI.

II. The Weight of a Cube

With usual terminology, a literal y_i is a variable x_i in direct or complemented form. Products are ANDs of literals. A product $q = y_{i_1}y_{i_2}...y_{i_k}$, $1 \leq k \leq n$, represents a cube of dimension $d(q) = n-k$, i.e., a cube of 2^{n-k} points in $\{0,1\}^n$. The intersection $p = p_1 \cap p_2$ of two cubes $p_1 = y_{i_1}...y_{i_{k_1}}$, $p_2 = y_{j_1}...y_{i_{k_2}}$ is obviously obtained as the AND of the two corresponding products. The intersection p is empty if and only if there is a literal in p_1 that appears complemented in p_2. Otherwise p is a cube of dimension $d(p) = r$, with $r = n-(k_1+k_2-c)$, and c is the number of common literals in p_1 and p_2.

Take p_1, p_2 as above, and let p_1, p_2 partially overlap. The set of points of $p_2 \setminus p_1$ can be covered in different ways by a set of at least $k_1 - c$ disjoint cubes of dimensions $r, r+1, ..., n-k_2-1$. For $n = 6$, letting $k_1 = 5$, $k_2 = 3$, $c = 2$ we have $r = 0$ and $d(p_1) = 1$, $d(p_2) = 3$, i.e., the intersection contains 1 point, and the two cubes contain 2 and 8 points, respectively. Therefore, $p_2 \setminus p_1$ contains 7 points and can be covered with $5-2 = 3$ cubes of dimensions 0, 1, 2. For an other example, consider cubes A and B in Figure 1(a). The set $A \setminus B$ contains the minterms 0000, 0001, and 0100. The disjoint covers for these points are $\overline{x}_1\overline{x}_2\overline{x}_3 + \overline{x}_1x_2\overline{x}_3\overline{x}_4$ and $\overline{x}_1\overline{x}_3\overline{x}_4 + \overline{x}_1\overline{x}_2\overline{x}_3x_4$, both containing two cubes.

Now, if p_1 is selected into a DSOP, p_2 must be discarded and the points of $p_2 \setminus p_1$ must be covered with at least $k_1 - c$ disjoint cubes instead of one (the single p_2). Then $k_1 - c - 1$ is the number of extra cubes required by the DSOP. If the function f can be represented by a SOP containing only p_1 and p_2, the selection of p_1 into a DSOP requires a total of $k_1 - c + 1$ cubes. In particular if $k_1 - c = 1$ the intersection p covers exactly one half of the points of p_2 and $p_2 \setminus p_1$ is also a cube. Clearly the general situation will not be that simple as the starting SOP for f, to be transformed into a minimal DSOP, will consist of a collection of cubes overlapping in groups. Still we define a weight for each cube p_i equal to the minimum number of extra cubes that the selection of p_i would induce in all the cubes intersecting p_i. Formally, let a SOP for f consist of partially overlapping products $p_1, p_2, ..., p_s$. We pose:

Definition 1: Let a product p_i of k literals intersect the products $p_{i_1}, ..., p_{i_t}$, such that p_i and p_{i_j} have c_j common literals. Then $w(p_i/p_{i_j}) = k - c_j - 1$ is the *weight of* p_i *relative to* p_{i_j}, and $w(p_i) = \sum_{j=1}^{t} w(p_i/p_{i_j})$ is the *weight* of p_i. If p_i does not intersect any other product, set $w(p_i) = -1$.

Thus, when p_i intersects p_{i_j}, the weight of p_i relative to p_{i_j} is the minimum number of additional products that we would have in the cover keeping p_i and covering p_i/p_{i_j} with non-overlapping products.

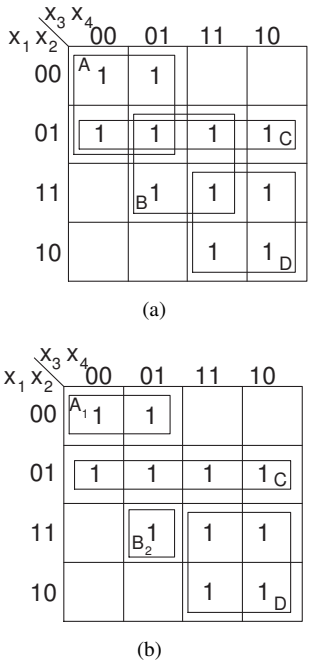

Fig. 1. (a) A minimal SOP of four cubes of dimension 2 in $\mathcal{B}^4$, with weights $w(A) = 1$, $w(B) = 2$, $w(C) = 0$, $w(D) = 1$. (b) A corresponding DSOP.

As an example, consider the function f of four variables, represented in Figure 1(a). A minimal SOP of f contains four cubes $A = \overline{x}_1\overline{x}_3$, $B = x_2x_4$, $C = \overline{x}_1x_2$, $D = x_1x_3$, all of dimension two. The weights are computed as follows. For A: $w(A/B) = 1$ (in fact, selecting A in a DSOP would require to covering the remaining three points of B with at least two disjoint cubes); $w(A/C) = 0$ (the residual two points of C can be covered with one cube); then $w(A) = 1$. For B: $w(B/A) = 1$; $w(B/C) = 0$; $w(B/D) = 1$; then $w(B) = 2$. For C: $w(C/A) = 0$; $w(C/B) = 0$; then $w(C) = 0$. For D: $w(D/B) = 1$; then $w(D) = 1$. As we shall explain in the next section, we start the construction of a DSOP by selecting the cubes with low weight and high dimension, breaking on the fly the ones that intersect a selected cube. In the present example, start by selecting C and reduce A and B to two subcubes A_1, B_1 of two points each. Then select D and further reduce B_1 to B_2 of one point. Then select A_1 and B_2, as shown in the DSOP of Figure 1(b). During the process the weights are updated as explained below.

III. DSOP synthesis algorithms

Let us consider an incompletely defined Boolean function $f : \{0,1\}^n \to \{0,1,-\}$ represented with a set of cubes $C = (C_{on}, C_{dc})$, where C_{on} covers the on set of f, i.e., the points v in $\{0,1\}^n$ such that $f(v) = 1$, and C_{dc} covers the don't care set of f, i.e., the points v in $\{0,1\}^n$ such that $f(v) = -$.

The new heuristic for DSOP construction uses four basic procedures working on an explicit representation of cubes. The first procedure BUILD-SOP(C, P) works on a set C of cubes covering an arbitrary function as above, to build a minimal (or quasi minimal) SOP P for that function. Note that, during the process, BUILD-SOP may be called on different sets C

```
algorithm DSOP(C, D)
INPUT:  A set of cubes C covering a function f
OUTPUT: A set of disjoint cubes D covering the function f

D = ∅
while (C ≠ ∅)
    BUILD-SOP(C, P)
    A = {d ∈ P | ∀c ∈ P \ {d} :  d ∩ c = ∅}
    D = D ∪ A
    P = P \ A
    WEIGHT(P)
    SORT(P)
    B = ∅
    while (P ≠ ∅)
        let p be the first element of P
        P = P \ {p}
        D = D ∪ {p}
        forall q ∈ P : p ∩ q ≠ ∅
            P = P \ {q}
            BREAK(q, p, Q)
            OPT(q, Q, P, B)
        forall r ∈ B : p ∩ r ≠ ∅
            B = B \ {r}
            BREAK(r, p, Q)
            B = B ∪ Q
    C = B
```

Fig. 2. The general algorithm for DSOP synthesis.

emerging in the computation. As a limit the cubes of C may be minterms, i.e., cubes of dimension 0. The second procedure WEIGHT(P) builds the weights for the cubes of a set P.

The third procedure SORT(P) sorts a set P of weighted cubes. This procedure comes in two versions: i) the cubes are ordered for decreasing dimension and, if the dimension is the same, for increasing weight; ii) the cubes are ordered for increasing weight and, if the weight is the same, for decreasing dimension. If two or more cubes have same weight and same dimension, their order is chosen arbitrarily. The two versions of SORT give rise to two different alternatives of the overall algorithm.

The forth procedure BREAK(q, p, Q) works on the set difference $q \setminus p$ between two cubes, to build an arbitrary minimal set Q of disjoint cubes covering $q \setminus p$. Note that this operation is easy since $q \setminus p$ can be obtained as $q \setminus (p \cap q)$, where the latter is the set difference between two cubes, i.e., q and $p \cap q$, in turn a cube because is the intersection of two cubes.

In practice, for BUILD-SOP one can use any minimization procedure (in our experiments we have used procedure ESPRESSO-NON-EXACT of the ESPRESSO suite [16]). Procedures WEIGHT and SORT (both versions) are obvious. Procedure BREAK is the one suggested in [7] and [12] as DISJOINT-SHARP.

In the overall process we consider four sets of cubes C, P, B, D. At the beginning C contains the cubes defining f, while P, B, D are empty. During the process C contains the cubes defining the part of f still to be covered with a DSOP; P contains the cubes of a SOP under processing; B temporarily contains cubes produced by BREAK as fragmentation of cubes of P; and D contains the cubes already assigned to the DSOP solution and, at the end, the solution itself.

The algorithms of our family share the structure shown in Figure 2 (its behaviour on incompletely specified functions is discussed at the end of this section). As long as f has not been completely covered with disjoint cubes, i.e., there are still cubes in the set C, a minimal (or quasi-minimal) SOP P for the part of f still to be covered is computed by the procedure BUILD-SOP. All cubes that do not intersect any other cube in P are removed from P and inserted in the DSOP D under construction; the remaining cubes are weighted and sorted. Then, the first cube p is extracted from P and inserted in the solution D. Each cube $q \in P$ that intersects p is removed from P, and a SOP Q for the set difference $q \setminus p$ is computed by the procedure BREAK. During this phase an optional optimization procedure OPT is called to decide how to handle the fragments in Q; depending on this optimization phase, different variants of the heuristic can be defined. Note that, since the points of p cannot be covered by any other cube, all fragments r already inserted in B must be tested for intersection with p and, if necessary, replaced with the SOP computed by BREAK for the set difference $r \setminus p$.

When P becomes empty, the fragments in B are moved to the set C and the algorithm iteratively builds a new SOP P covering the points that are not yet covered by the DSOP D under construction. The iterations terminate when C becomes empty.

We have designed and tested five variants of our heuristic based on five different versions of the optimization procedure OPT, with different degrees of sophistication. The first variant, **DSOP-1**, is the simplest, and computationally fastest, as OPT simply inserts the cubes of Q into the set of fragments B:

```
in DSOP-1:
    procedure OPT(q, Q, P, B)
        B = B ∪ Q
```

Example 1: For an example, Figure 1(b) shows a DSOP form for the SOP form of Figure 1(a), computed by algorithm DSOP-1. At the beginning $D = \emptyset$ and $P = \{\overline{x}_1x_2, x_1x_3, \overline{x}_1\overline{x}_3, x_2x_4\}$, sorted for decreasing dimensions of cubes and then for increasing weighs (we recall that, $w(\overline{x}_1x_2) = 0$, $w(x_1x_3) = 1$, $w(\overline{x}_1\overline{x}_3) = 1$, and $w(x_2x_4) = 2$). The first cube considered is $p = \overline{x}_1x_2$, which is removed from P and inserted in D. Its intersecting cubes, $\overline{x}_1\overline{x}_3$ and x_2x_4, are then broken generating the residuals cubes $\overline{x}_1\overline{x}_2\overline{x}_3$ and $x_1x_2x_4$, respectively, which are inserted in B, while $\overline{x}_1\overline{x}_3$ and x_2x_4 are removed from P. The last cube in P to be considered is then x_1x_3 that is inserted directly in D, since there are not any other remaining cubes in P. Its intersecting cube $x_1x_2x_4$ in B is then reduced to $x_1x_2\overline{x}_3x_4$. The second **while** $(P \neq \emptyset)$ iteration starts with $P = \{\overline{x}_1\overline{x}_2\overline{x}_3, x_1x_2\overline{x}_3x_4\}$ and $D = \{\overline{x}_1x_2, x_1x_3\}$, and terminates with the final DSOP $D = \{\overline{x}_1x_2, x_1x_3, \overline{x}_1\overline{x}_2\overline{x}_3, x_1x_2\overline{x}_3x_4\}$.

In the second variant, **DSOP-2**, after a cube p has been selected and moved to D, each cube q intersecting p is, as before, fragmented and moved to B. In addition the optimization procedure updates the weight of all cubes $r \in P$ that intersect q, and then sorts the cubes in P again:

in DSOP-2:
procedure OPT(q, Q, P, B)
$B = B \cup Q$
$I = \{r \in P \mid q \cap r \neq \emptyset\}$
WEIGHT(I)
SORT(P)

A disadvantage of both versions is that whenever a cube p is moved from P to D, all cubes q intersecting p are fragmented and removed from the set P. Hence, the fragments, even the big ones, are "out of the game" and cannot participate in the construction of the DSOP D until P becomes empty and a new SOP covering all fragments in the set B is computed. Consequentially, small cubes in P could be selected first, possibly damaging the quality of the final result, i.e., the size of the final DSOP. To partially avoid this disadvantage, we have implemented a third version of the heuristic, **DSOP-3**, in which whenever a cube $p \in P$ is moved to D, each cube q intersecting p is, as before, fragmented and moved to B, and, in addition, all cubes $r \in P$ intersecting q are moved to B as well:

in DSOP-3:
procedure OPT(q, Q, P, B)
$I = \{r \in P \mid q \cap r \neq \emptyset\}$
$B = B \cup Q \cup I$
$P = P \setminus I$

In this way, the cubes of P intersecting the fragments already in B cannot be selected,while is avoided the possible fragmentation of big cubes in B. Moreover, we leave open the possibility of selecting these big cubes in the next iterations of the algorithm, This version of the heuristic is computationally more expensive, since in the internal while loop less cubes can be selected (P empties faster), and procedure BUILD-SOP must be executed more frequently.

The fourth version of the heuristic, **DSOP-4**, checks whether the set Q contains only one fragment, i.e., $q \setminus p$ is a cube. In this case, this only fragment is put back in P. The cubes left in P are then weighted and sorted again:

in DSOP-4:
procedure OPT(q, Q, P, B)
if ($|Q| = 1$)
$P = P \cup Q$
else
$B = B \cup Q$
WEIGHT(P)
SORT(P)

Finally, in the last version of the heuristic that we have tested, **DSOP-5**, the biggest fragment in the set Q is always put back in P. The cubes left in P are then weighted and sorted again. In this way, big fragments remain part of the game in the present iteration of the algorithm:

in DSOP-5:
procedure OPT(q, Q, P, B)
let b be the biggest cube in Q
$P = P \cup \{b\}$
$B = B \cup Q \setminus \{b\}$
WEIGHT(P)
SORT(P)

The performances of these five procedures are discussed in Section V. We have observed experimentally that more sophisticated optimization procedures do not always provide better quality results. Experimental results have also outlined how the BUILD-SOP procedure, i.e., fre-synthesizing the remaining cubes, seems to be crucial for obtaining compact DSOPs.

Let us now briefly consider the case of the DSOP synthesis of incompletely specified Boolean functions. Our heuristic does not consider explicitly the presence of don't cares; indeed, the first call of the BUILD-SOP procedure produces a SOP P covering the whole on set of f and a subset of its don't care set. Then, the algorithm works on the SOP P, treating all points covered by its cubes as if they belonged to the on set of f, i.e., there is no distinction between points originally in the on set of f and points originally in the don't care set. In particular, the successive calls of BUILD-SOP on the part of f still to be covered with a DSOP, treat the function as if it were completely specified. Of course, each cube in the SOP P computed by the first call of BUILD-SOP covers at least one point in the on set of f, as cubes covering only points in the don't care set are discarded by the SOP minimization algorithm. However, the final disjoint cover D for f could contain cubes covering only points originally in the don't care set. In fact, cubes in D are either entire cubes of the starting SOP P, or sub-cubes of cubes in P (besides new cubes and sub-cubes originated by the successive calls of BUILD-SOP) and some sub-cubes (or new cubes) could only cover don't care points. Thus, all versions of our heuristic could be improved checking whether a cube p contains only points in the don't care set of the function f, before adding it to the DSOP solution D under construction. Unfortunately, such a check can be computationally expensive, and for this reason we have not added it as a "default" procedure in our algorithm.

We conclude with an observation concerning the complexity of the DSOP minimization problem. First of all, note that the complexity of the proposed heuristic is polynomial in the size of the output, i.e., in the number of products of the computed DSOP form. As the standard SOP minimization is as complex as set covering [6], [15], DSOP minimization can be compared to the set partitioning (or minimal exact cover) problem. The minimal exact cover problem can be described as follows: given a family of subset S of a set U and a positive integer k, is there a subset family $T \subseteq S$ such that the subsets in T are k in number, are disjoint, and their union is the entire set U? The minimal exact cover is NP-hard since it can be easily reduced by the "exact cover problem" introduced by Karp in 1972 [8] setting k to the cardinality of U.

algorithm PARTIAL-BREAK($q, p, sopD, sopS, Q, R$)
INPUT: The chosen cube p, the cube q that can be broken, the two SOPs $sopD$ and $sopS$ whose union represents f
OUTPUT: A minimal set Q of disjoint cubes covering $q \setminus p$ and a set R of the points of $q \setminus p$ that can be covered more than once

$R = \emptyset$
$p_i = q \cap p$
if ($p_i \subseteq sopS$) *// all points of p_i can be covered more than once*
  $Q = \emptyset$
else if ($p_i \subseteq sopD$) *// all points of p_i must be covered once*
  $Q =$ DISJOINT_SHARP(q, p_i)
else *// p_i intersects both $sopD$ and $sopS$*
  $Q =$ DISJOINT_SHARP(q, p_i)
  $R = p_i \cap sopS$

Fig. 3. The procedure PARTIAL-BREAK to be used in partial DSOP synthesis.

IV. Partial DSOP synthesis

As mentioned in Section I, some applications require to cover some minterms of a Boolean function exactly once while other minterms can be covered more than once. A typical example, is when the points in the on set of a function must be covered exactly once, while the points in the don't care set can be covered more than once [9], [5]. In this section we present a general heuristic to efficiently compute a *partial DSOP* cover.

The heuristic makes use of two sums of products as input. The first SOP, $sopD$, contains all points of the on and don't care set of the function f that must be covered only once (DSOP part), while the second SOP, $sopS$, contains all the points of f that can be covered more than once (SOP part). These two SOPs are disjoint. The output of the heuristic is a cover of the overall function f, represented by the union of the two SOPs $sopD$ and $sopS$ that respects the specifications. Note that when $sopD$ is empty the problem is a classical SOP minimization, while when $sopS$ is empty the problem is a classical DSOP minimization.

The algorithm uses four basic procedures as for the DSOP synthesis of Section III. In particular BUILD-SOP, WEIGHT, and SORT are the same. The forth procedure PARTIAL-BREAK($q, p, sopD, sopS, Q, R$) works on the set difference $q \setminus p$ between two cubes, to build an arbitrary minimal set Q of disjoint cubes covering $q \setminus p$, if $q \cap p$ is not entirely contained in $sopS$. If $q \cap p$ is contained in $sopS$, the cube q is not broken and we can keep it in the set P which contains the cubes to be considered in the current iteration. In this case we then set $Q = \emptyset$. Moreover, the procedure PARTIAL-BREAK builds a set R containing points of $q \setminus p$ that can be covered more than once and can therefore be added as don't cares to C. In this way, these points, that have been already covered, could be used again in the minimization phase to get a smaller cover. This procedure, different from the one used for DSOP synthesis, is presented in Figure 3.

The overall minimization heuristic is presented in Figure 4. As for the DSOP synthesis, the heuristic makes use of four sets of cubes C, P, B, D. At the beginning $C = sopD \cup sopS$ contains the cubes defining f while P, B, D are empty. During

algorithm PARTIAL-DSOP($sopD, sopS, D$)
INPUT: Two disjoint SOPs describing the points of f that must be covered only once ($sopD$) and the points of f that can be covered more than once ($sopS$)
OUTPUT: A partial DSOP D for the function f

$C_{on} = sopD_{on} \cup sopS_{on}$
$C_{dc} = sopD_{dc} \cup sopS_{dc}$
while ($C_{on} \neq \emptyset$)
  BUILD-SOP(C, P)
  $A = \{d \in P \mid \forall c \in P \setminus \{d\} : \ d \cap c = \emptyset\}$
  $D = D \cup A$
  $P = P \setminus A$
  WEIGHT(P)
  SORT(P)
  $B = \emptyset$
  while ($P \neq \emptyset$)
    let p be the first element of P
    $P = P \setminus \{p\}$
    $D = D \cup \{p\}$
    forall $q \in P : \ p \cap q \neq \emptyset$
      PARTIAL-BREAK($q, p, sopD, sopS, Q, R$)
      if ($Q \neq \emptyset$) $P = P \setminus \{q\}$
      OPT(q, Q, P, B)
      $C_{dc} = C_{dc} \cup R$
    forall $r \in B : \ p \cap r \neq \emptyset$
      PARTIAL-BREAK($r, p, sopD, sopS, Q, R$)
      if ($Q \neq \emptyset$) $B = B \setminus \{r\}$
      $B = B \cup Q$
      $C_{dc} = C_{dc} \cup R$
  $C_{on} = B$

Fig. 4. Algorithm for partial DSOP synthesis.

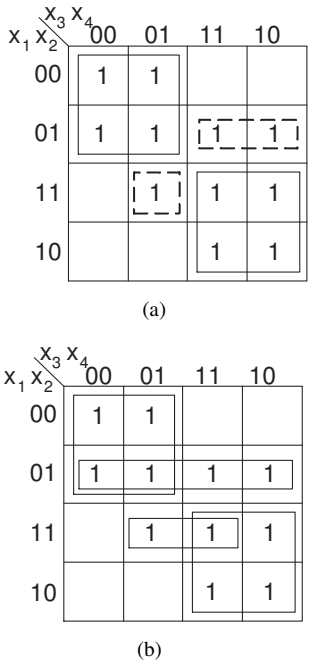

Fig. 5. (a) sopS (cubes with solid lines) and sopD (cubes with dotted lines). (b) A corresponding partial DSOP.

the processing C contains the cubes defining the part of f still to be covered with a partial DSOP. P contains the cubes of a SOP under processing. B temporarily contains cubes produced

SORT VERSION: dimension/weight													
			SOP	DSOP-1		DSOP-2		DSOP-3		DSOP-4		DSOP-5	
Bench	**in**	**out**	**size**	**size**	**time**	**size**	**time**	**size**	**time**	**size**	**time**	**size**	**time**
accpla	50	69	175	1457	11.68	1458	13.42	1190	10.55	**1125**	5.61	1528	21.60
addm4	9	8	200	218	0.26	221	0.31	**214**	0.40	222	0.19	224	0.19
alu4	14	8	575	923	1.51	921	2.00	**881**	2.32	1051	3.36	1044	2.87
apex3	54	50	280	**345**	0.62	**345**	0.65	350	0.68	366	2.04	400	2.11
apex4	9	19	436	506	0.34	506	0.36	503	0.31	502	0.52	**501**	0.59
b2	16	17	106	131	0.42	131	0.49	**121**	0.54	130	0.93	127	0.56
bc0	26	11	179	214	0.47	214	0.53	**202**	0.68	212	0.53	208	0.51
chkn	29	7	140	187	0.87	187	0.88	**168**	0.88	215	0.48	221	0.42
clip	9	5	120	151	0.28	150	0.29	**140**	0.39	153	0.20	157	0.16
cps	24	109	163	**184**	0.75	**184**	0.76	204	0.89	219	0.38	225	0.38
dist	8	5	123	135	0.22	135	0.23	130	0.38	**128**	0.15	129	0.16
ex5	8	63	74	126	0.79	126	0.44	**122**	0.80	137	0.39	141	0.48
gary	15	11	107	134	0.52	134	0.35	**124**	0.49	126	0.28	127	0.16
ibm	48	17	173	366	1.23	366	0.59	**361**	0.99	373	0.46	391	0.30
in4	32	20	212	312	1.36	312	0.84	**280**	1.33	303	0.54	304	0.56
intb	15	7	631	811	2.03	818	1.61	**798**	2.57	922	2.56	952	2.91
jbp	36	57	122	135	0.57	135	0.26	**127**	0.43	134	0.20	136	0.17
mainpla	27	54	172	296	4.67	296	3.00	293	3.23	288	5.37	**260**	5.20
max1024	10	6	274	**332**	0.32	334	0.33	334	0.54	347	0.35	345	0.32
misex3	14	14	690	1070	2.72	1073	1.49	**1032**	2.68	1159	2.57	1309	3.48
soar	83	94	353	447	1.93	447	1.30	**434**	1.58	442	0.59	456	0.58
sym10	10	1	210	232	0.43	**231**	0.51	232	1.11	235	1.01	245	0.85
table3	14	14	175	181	0.41	181	0.23	180	0.33	**179**	0.16	**179**	0.18
table5	17	15	158	167	0.39	167	0.36	**161**	0.38	**161**	0.24	**161**	0.25
tial	14	8	581	943	1.78	937	1.96	**874**	2.98	1071	2.84	1040	2.50
vtx1	27	6	110	**204**	0.45	**204**	0.49	**204**	0.64	208	0.34	213	0.31
x7dn	66	15	538	796	1.30	**784**	1.43	812	1.57	813	0.88	864	0.76

TABLE I
COMPARISON OF FIVE DIFFERENT VARIANTS OF THE DSOP MINIMIZATION HEURISTIC (SORT VERSION: **dimension/weight**.) THE SIZE OF THE BEST DSOP REPRESENTATION COMPUTED FOR EACH BENCHMARK IS IN BOLDFACE.

by BREAK as fragmentation of cubes of P. D contains the cubes already assigned to the partial DSOP solution and, at the end, the solution itself. OPT(q, Q, P, B) is an optional optimization procedure to decide how to handle the fragments produced by the procedure BREAK. As before, depending on this optimization phase, different variants of the heuristic can be defined.

Example 2: Consider the function shown in Figure 5(a). Suppose that $sopD = \{\overline{x}_1x_2x_3, x_1x_2\overline{x}_3x_4\}$ (cubes with dotted lines in the figure) and $sopS = \{\overline{x}_1\overline{x}_3, x_1x_3\}$ (cubes with solid lines). A partial DSOP for f is shown in Figure 5(b). This expression is obtained with the partial DSOP algorithm as described in the following. Let OPT(q, Q, P, B) be the simple command $B = B \cup Q$ (as in the **DSOP-1** procedure). At the beginning $D = \emptyset$ and, after the SOP minimization phase, $P = \{\overline{x}_1x_2, x_1x_3, \overline{x}_1\overline{x}_3, x_2x_4\}$, sorted for decreasing dimensions of cubes and then for increasing weighs (note that we have the same initial P of Example 1). The first cube $p = \overline{x}_1x_2$ is removed from P and inserted in D. Its intersecting cubes are $\overline{x}_1\overline{x}_3$ and x_2x_4. In the procedure PARTIAL-BREAK, the intersection between x_2x_4 and $\overline{x}_1x_2$ is $p_i = \overline{x}_1x_2x_4$. Note that p_i intersects both $sopD$ and $sopS$, thus $Q = \{x_1x_2x_4\}$ and $R = \{\overline{x}_1x_2\overline{x}_3x_4\}$ (i.e., $x_1x_2x_4$ and $\overline{x}_1x_2\overline{x}_3x_4$ will be inserted in B and in the don't care set of C, respectively). Moreover, we compute the intersection p_i between $\overline{x}_1\overline{x}_3$ and $\overline{x}_1x_2$, obtaining $\overline{x}_1x_2\overline{x}_3$ which is entirely contained in $sopS$. Thus, in this case $Q = R = \emptyset$, then $\overline{x}_1\overline{x}_3$ is not broken and it is not removed from P. Similar operations are performed on $\overline{x}_1\overline{x}_3$, and on $x_1x_2x_4$ contained in B. The second **while** $(P \neq \emptyset)$ iteration, which starts with the $P = \{x_1x_2x_4\}$ and $D = \{\overline{x}_1x_2, x_1x_3, \overline{x}_1\overline{x}_3\}$, terminates with the partial DSOP shown in Figure 5(b).

V. EXPERIMENTAL RESULTS

In this section we present and discuss the results obtained with the heuristics presented above to the standard ESPRESSO benchmark suite [16]. All experiments were performed on a 1.8 GHz PowerPC with 1 GB of RAM.

A. DSOP synthesis

We have considered the five different variants of the heuristic described in Section III, denoted as **DSOP-1, DSOP-2, DSOP-3, DSOP-4, DSOP-5**. For each variant, we have run both versions of the procedure SORT, to estimate the practical effectiveness of each version. Namely we have ordered the cubes for decreasing dimension and, in case of equal dimension, for increasing weight (version **dimension/weight**). Then we have ordered the cubes for increasing weight and, in case of equal weight, for decreasing dimension (version **weight/dimension**).

Since the benchmarks are multi-output functions and the algorithm is described for single output function, in the experiments we have considered each output separately, but the minimization phase with ESPRESSO is performed in a multi-output way. Moreover, common disjoint cubes of several output are counted only once.

Tables I and II report a significant subset of the experiments. In particular, Table I reports the performances of the heuristics

SORT VERSION: weight/dimension													
			SOP	DSOP-1		DSOP-2		DSOP-3		DSOP-4		DSOP-5	
Bench	**in**	**out**	**size**	**size**	**time**	**size**	**time**	**size**	**time**	**size**	**time**	**size**	**time**
accpla	50	69	175	1779	24.57	1717	32.46	**1317**	16.80	1535	34.29	3078	97.73
addm4	9	8	200	220	0.26	222	0.27	**217**	0.29	222	0.20	223	0.13
alu4	14	8	575	1138	2.25	**1065**	2.12	1276	4.74	1269	5.23	1211	3.69
apex3	54	50	280	**337**	0.54	342	0.57	347	0.6	356	0.82	386	0.94
apex4	9	19	436	506	0.34	506	0.36	503	0.33	502	0.38	**500**	0.25
b2	16	17	106	131	0.41	131	0.44	**120**	0.62	126	0.36	125	0.36
bc0	26	11	179	230	0.60	218	0.53	**210**	1.18	218	0.44	211	0.40
chkn	29	7	140	598	3.67	448	3.55	**216**	2.86	386	2.21	426	1.70
clip	9	5	120	154	0.32	153	0.29	**143**	0.38	157	0.2 0	159	0.17
cps	24	109	163	**184**	0.75	**184**	0.88	204	1.19	217	0.36	223	0.36
dist	8	5	123	138	0.21	138	0.23	**133**	0.36	134	0.15	**133**	0.15
ex5	8	63	74	128	0.38	**125**	0.44	142	0.77	141	0.39	148	0.31
gary	15	11	107	139	0.28	131	0.27	132	0.43	**124**	0.15	125	0.15
ibm	48	17	173	431	0.69	**393**	0.64	416	1.24	415	0.56	478	0.47
in4	32	20	212	329	0.72	331	0.81	321	1.29	**303**	0.52	319	0.48
intb	15	7	631	955	1.79	**932**	1.83	1125	3.65	1130	4.99	1173	3.84
jbp	36	57	122	151	0.35	147	0.36	**128**	0.33	140	0.16	147	0.17
mainpla	27	54	172	459	2.86	405	2.47	387	3.33	366	2.76	**338**	2.23
max1024	10	6	274	334	0.30	330	0.36	**324**	0.50	339	0.34	338	0.29
misex3	14	14	690	**1132**	1.28	1155	1.75	1317	4.58	1234	3.67	1464	4.36
soar	83	94	353	451	1.16	449	1.25	**430**	1.41	440	0.61	464	0.60
sym10	10	1	210	**233**	0.42	234	0.48	248	1.37	239	1.19	258	1.38
table3	14	14	175	181	0.21	181	0.24	180	0.24	**179**	0.14	**179**	0.14
table5	17	15	158	167	0.31	167	0.32	**161**	0.28	**161**	0.20	**161**	0.17
tial	14	8	581	1121	2.22	**1060**	2.13	1371	6.68	1330	5.25	1322	3.65
vtx1	27	6	110	**236**	0.50	247	0.51	258	0.93	313	0.63	317	0.62
x7dn	66	15	538	1078	1.89	1010	2.23	**919**	2.80	1068	2.32	1043	1.30

TABLE II
COMPARISON OF FIVE DIFFERENT VARIANTS OF THE DSOP MINIMIZATION HEURISTIC (SORT VERSION: **weight/dimension**.) THE SIZE OF THE BEST DSOP REPRESENTATION COMPUTED FOR EACH BENCHMARK IS IN BOLDFACE.

with respect to the first version of the SORT procedure, while Table II is relative to the second SORT procedure. All benchmarks in these tables are completely specified. In both tables, the first column reports the name of the benchmark; the following two columns give the number of inputs and outputs; the column labeled **SOP** shows the number of products in a SOP representation computed by ESPRESSO in the heuristic mode; finally the remaining five pairs of columns report the number of disjoint products in the DSOP expressions computed by our heuristics and the corresponding synthesis time.

			DSOP-3 (a)		DSOP-3 (b)	
Bench	**in**	**out**	**size**	**time**	**size**	**time**
b10	15	11	115	0.49	115	17.04
b3	32	20	279	1.21	279	47.34
bca	26	46	189	0.29	189	49.54
bcb	26	39	162	0.26	162	42.33
bench1	9	9	250	0.32	**210**	14.92
ex1010	10	10	876	1.34	**665**	73.00
exam	10	10	145	0.33	**107**	62.26
exep	30	63	130	0.53	**120**	8.33
exps	8	38	151	0.31	151	37.91
pdc	16	40	381	0.98	**277**	37.14
spla	16	46	347	0.64	347	37.06
test2	11	35	2322	2.40	**2054**	324.84
test3	10	35	1462	1.81	**1204**	159.60

TABLE III
DSOP SYNTHESIS OF INCOMPLETELY SPECIFIED BENCHMARKS, WITHOUT (**DSOP-3 (a)**) AND WITH (**DSOP-3 (b)**) ELIMINATION OF CUBES COVERING ONLY DON'T CARES. THE SIZE OF THE BEST DSOP IS IN BOLDFACE.

As Table I and Table II clearly show, the third variant of the heuristic, together with the first version of procedure SORT (version **dimension/weight**), gives the best results regarding the size of the resulting DSOP forms, and its running times are comparable to those of the other variants, and sometimes even lower.

We have then tested the performances of the best variant of our heuristic on incompletely specified benchmarks. Table III reports a subset of our experiments. We have run the heuristic without the elimination of cubes covering only don't cares points from the solution under construction (**DSOP-3 (a)**), and with such elimination (**DSOP-3 (b)**). As the table clearly shows, the elimination of these cubes naturally produces better solutions in terms of size, but the computational time is much higher.

In another series of experiments we compared our heuristic (with the third version of the optimization phase, and without elimination of cubes of don't cares only) with other DSOP minimization methods. We considered two techniques working, as ours, on explicit representation of cubes, and one method based on binary decision diagrams. The first algorithm [3] sorts cubes in a minimal SOP according to their size, and compares the largest cube with all the others, starting from the smallest ones. In the next step, the second largest cube is selected and compared to all smaller ones, etc. As a last step, the cubes are merged wherever possible. The second algorithm, presented in [13], exploits the property of the most binate variable in a set of cubes to compute a DSOP form. Finally, the third approach, presented in [4], makes use of

Bench	in	out	PLA	SOP	DSOP ESPR.	DSOP [3]	DSOP [13]	DSOP [4]	DSOP-3
5xp1	7	10	75	65	99	70	–	82	70
9sym	9	1	87	86	209	166	148	148	134
alu4	14	8	1028	575	3551	–	–	1545	881
b12	15	9	431	43	691	57	–	60	51
clip	9	5	167	120	359	162	–	262	140
co14	14	1	47	14	14	–	14	14	14
max1024	10	6	1024	274	775	–	–	444	334
misex1	8	7	32	12	18	15	–	34	15
misex2	25	18	29	28	29	28	–	30	28
mlp4	8	8	256	128	206	–	–	203	143
rd53	5	3	32	31	31	31	–	35	31
rd73	7	3	141	127	127	127	–	147	127
rd84	8	4	256	255	255	–	–	294	255
sym10	10	1	837	210	367	–	240	240	232
t481	16	1	481	481	2139	–	2139	1009	841
x7dn	66	15	622	538	1697	–	–	1091	812
xor5	5	1	16	16	16	–	16	16	16

TABLE IV
COMPARISON WITH OTHER TECHNIQUES

BDDs, exploiting the efficiency resulting from the implicit representation of the products. Observe in fact that a DSOP form can be extracted in a straightforward way from a BDD, as different one-paths correspond to disjoint cubes.

Table IV reports a cost-oriented comparison among the different methods. The first three columns are as before. Columns four and five report the number of products in the PLA realization and in the SOP form heuristically minimized by ESPRESSO in the heuristic mode. The column labeled **DSOP** ESPR. shows the size of the DSOP computed running ESPRESSO with the option "-Ddisjoint" on the previously computed SOP form. The next three columns report the sizes, when available, of the DSOP forms computed with the methods discussed in [3], [13], and [4], respectively. Finally, the last column shows the size of the DSOPs computed with our heuristic (third variant).

As the table clearly shows, our method always generates smaller DSOP representations, and the gain in size can be quite striking, as for instance for the benchmarks *alu4, clip,* and *t481*. Note that, even considering the other variants of the heuristic, and the different sorting criterion, our technique almost always compares favorably. A time comparisons among all these different methods was not possible due to the partial absence of CPU times specification in the literature.

B. Partial DSOP synthesis

In order to test our partial DSOP synthesis algorithm, we have applied the heuristic to the classical ESPRESSO benchmark suite [16] with the following meaning. We have considered only benchmarks with don't cares, where the on set of the benchmark is the on set of $sopD$, and the don't care set of the benchmark is the don't care set of $sopS$.

Table V reports a subset of our experimental results. The column labeled **SOP** shows the number of products in a SOP representation computed by ESPRESSO in the heuristic mode. The remaining three pairs of columns report the number of products and the corresponding synthesis time for the following three forms (all computed with the third version of the optimization phase, the dimension/weight sort version, and with the elimination of cubes covering don't cares only):

1) **DSOP**: a DSOP for the original function, with the choice of don't cares performed by ESPRESSO in the heuristic mode. Each don't care point is covered **at most** once.
2) **P-DSOP (a)**: a partial DSOP for the original function, with the choice of don't cares performed by ESPRESSO in the heuristic mode. Don't care points are either eliminated or covered **at least** once.
3) **P-DSOP (b)**: a partial DSOP for the original function, where all the don't cares of the function are in play (they have all been covered during the first SOP minimization). Don't care points are either eliminated or covered **at least** once in the final form.

Note that the results in the column **SOP** are better than ours because the resulting form is not disjoint.

The table suggests that the best solution is the one relative to the choice of don't cares made by ESPRESSO. Moreover, it appears clearly from these results that the option of covering more than once the don't care points of the function (**DSOP-3 (a)**) gives better results, especially for big benchmarks.

VI. CONCLUSIONS AND FUTURE WORK

Although deriving an optimal DSOP representation of a Boolean function is a hard problem, we have described a heuristic that has been implemented and tested.

From the experimental results we conclude that exploiting SOP minimization for DSOP synthesis is a crucial idea. In fact, comparing our results with the ones in the literature we always obtain equal or smaller forms. We observe that the fact that SOP and DSOP problems are so close is not intuitive. In fact, we would have expected that efficient strategies to solve the two problems would be different since DSOP minimization appears to be much harder then SOP synthesis. Nevertheless, the experiments show that, starting from minimal or quasi-minimal SOP expressions, we can heuristically derive very

Bench	in	out	SOP size	DSOP size	DSOP time	P-DSOP (a) size	P-DSOP (a) time	P-DSOP (b) size	P-DSOP (b) time
alu3	10	8	66	67	4.65	67	10.83	67	13.06
apla	10	12	25	33	4.11	**25**	2.49	**25**	13.204
b10	15	11	100	**115**	17.04	**115**	15.96	117	18.22
b3	32	20	211	279	47.34	279	51.39	279	54.80
b4	33	23	54	62	10.06	62	5.99	62	7.74
bca	26	46	180	**189**	49.54	**189**	33.63	190	53.11
bcb	26	39	155	162	42.33	162	29.08	162	42.18
bcc	26	45	137	145	26.08	145	31.19	145	43.24
bcd	26	38	117	121	17.30	121	20.51	121	29.14
bench1	9	9	139	210	14.92	**164**	18.30	246	86.90
dk17	10	11	18	22	1.32	**19**	1.50	**19**	15.57
dk27	9	9	10	12	1.04	**10**	0.73	**10**	10.15
dk48	15	17	22	24	1.17	**22**	1.40	**22**	69.59
duke2	22	29	12	26	3.06	24	4.10	**23**	26.16
ex1010	10	10	284	665	73.00	**481**	87.24	739	282.44
exam	10	10	67	107	62.26	**89**	66.78	168	115.78
exp	8	18	59	72	7.11	70	7.28	**63**	16.25
exps	8	38	136	**151**	37.91	**151**	41.37	152	5.61
inc	7	9	30	**37**	2.52	38	3.298	41	4.87
mark1	20	31	19	29	2.71	**23**	5.26	25	136.14
p1	8	18	55	90	7.51	**67**	12.26	79	32.37
p3	8	14	39	71	4.77	**47**	10.29	52	18.27
pdc	16	40	145	277	37.14	203	68.65	**191**	291.51
sao2	10	4	9	24	1.09	20	4.01	**19**	6.86
spla	16	46	260	347	37.06	347	50.05	347	53.21
t2	17	16	53	**58**	3.34	59	5.36	59	8.48
t4	12	8	16	18	1.31	**17**	1.79	21	10.49
test1	8	10	121	169	10.52	**141**	12.92	197	45.38
test2	11	35	1103	2054	324.84	**1354**	361.73	1996	2921.75
test3	10	35	541	1204	159.60	**746**	179.11	1001	1405.99
test4	8	30	120	466	65.34	335	82.44	**278**	253.03
x1dn	27	6	70	148	12.51	**98**	26.31	106	48.30

TABLE V
PARTIAL DSOP SYNTHESIS, USING THE SUBSET OF DON'T CARES SELECTED BY ESPRESSO-NON-EXACT (**P-DSOP (a)**) OR ALL DON'T CARES OF THE ORIGINAL FUNCTION (**P-DSOP (b)**).

compact DSOP forms. Moreover, from Table V we also infer that the choice of the don't cares, which are used as ones of the function, performed for the SOP minimization is nearly always the best choice also for DSOP synthesis. Therefore, as a future work, it would be interesting to further study the closeness of SOP and DSOP minimal forms both in theoretical and experimental way.

Cubes are currently explicitly represented; it would be interesting to use implicit data structures (e.g., BDDs) and symbolically perform all the operations in the heuristic. In this way, we should speed up the computation, and perform operations like the elimination of cubes of don't cares in a more efficient way.

From a more theoretic perspective, it could be an interesting development to study more deeply the approximability properties of DSOP minimization, with the aim of designing approximation algorithms, instead of heuristics. Recall that a p-approximation algorithm for a minimization problem always yields solutions whose cost C is $\leq pC^*$, where C^* is the cost of an optimal solution [6]. Thus, both heuristics and approximation algorithms do not guarantee the minimality of their solutions, but an approximation algorithm guarantees near-optimum solutions, whereas we cannot perform any prediction on the result of a heuristic. As a first step in this direction we should understand when our heuristic returns a DSOP whose cost is much higher then the cost of an optimal DSOP.

REFERENCES

[1] A. Bernasconi, V. Ciriani, F. Luccio, and L. Pagli. A New Heuristic for DSOP Minimization. *Proc. 8th International Workshop on Boolean Problems (IWSBP)* (2008).

[2] B.J. Falkovsky and C.H. Chang. Paired Haar Spectra Computation Through Operations on Disjoint Cubes. *IEEE Proc. on Circuits, Devices, and Systems* (1999) 117-123.

[3] B.J. Falkovsky,I. Schäfer, and C.H. Chang. An Effective Computer Algorithm for the Calculation of Disjoint Cube Representation of Boolean Functions. *Proc. IEEE International Midwest Symp. on Circuits and Systems* (1993) 1308-1311.

[4] G. Fey and R. Drechsler. Utilizing BDDs for Disjoint SOP Minimization. *Proc. IEEE International Midwest Symp. on Circuits and Systems* 2 (2002) 306-309.

[5] P. Fišer. Personal communication. (2010).

[6] M.R. Garey and D.S. Johnson. Computer and Intractability: A Guide to the Theory of NP-completeness. *W.H. Freeman and Company* (1979).

[7] S.J. Hong, R.G. Cain and D.L. Ostapko. MINI: A Heuristic Approach for Logic Minimization. *IBM Journal of Research and Development* 18 (5) (1974) 443-458.

[8] R.M. Karp Reducibility Among Combinatorial Problems. in *R. E. Miller and J. W. Thatcher (editors): Complexity of Computer Computations*. New York: Plenum Press (1972), 85103.

[9] I. Lemberski and P. Fišer Multi-Level Implementation of Asynchronous Logic Using Two-Level Nodes *Proc. 4th IFAC Workshop on Discrete-Event System Design* (2009)

[10] T. Sasao. EXMIN2: A Simplification Algorithm for Exclusive-OR-Sum-of -Products Expression for Multiple-Valued-Input Two-Valued-Output functions. *IEEE Trans. on Computer Aided Design* 12 (1993) 621-632.

[11] T. Sasao. A Design Method for AND-OR-EXOR Three-Level Networks. *Proc. ACM/IEEE International Workshop on Logic Synthesis* (1995) 8:11-8:20.

[12] T. Sasao. Switching Theory for Logic Synthesis, *Kluwer Academic Publishers* (1999).
[13] L. Shivakumaraiah and M. Thornton. Computation of Disjoint Cube Representation Using a Maximal Binate Variable Heuristic. *Proc. Southeastern Symp. on System Theory* (2002) 417-421.
[14] M.A. Thornton, R. Drechsler, and D.M. Miller. *Spectral Techniques in VLSI CAD*. Kluwer Academic Publ., 2001.
[15] C. Umans, T. Villa, and A.L. Sangiovanni-Vincentelli. Complexity of Two-Level Logic Minimization. *IEEE Trans. on Computer-Aided Design* 25 (7) (2006) 1230-1246.
[16] S. Yang. *Synthesis on Optimization Benchmarks*. User guide, Microelectronic Center, 1991. Benchmarks at: ftp://ftp.sunsite.org.uk/computing/general/espresso.tar.Z.