FAST APPROXIMATE MINIMIZATION OF THREE-LEVEL MULTI-OUTPUT STRONGLY UNSPECIFIED BOOLEAN FUNCTIONS

Sanof Mohamed, Marek Perkowski +, and Lech Jozwiak †,
COMPASS Design Automation, 1865 Lundy Ave, M/S 430,
San Jose CA 95131, USA, Tel: 408-434-7893,
Fax 408-434-7981, sanof@compass-da.com,
+ Dept. of Electrical Engn., Portland State University,
Portland, Oregon 97207-0751, USA, Tel: 503-725-5411,
Fax: 503-725-4882, mperkows@ee.pdx.edu
† Faculty of Electrical Engineering, Eindhoven University of Technology,
P.O. Box 513, EH 10.25, 5600 MB Eindhoven, The Netherlands,
Tel: +31.40.2473645, Fax: +31.40.2433066, lech@eb.ele.tue.nl

Abstract— A CDEC gate, or a Condition Decoder, being a product of an AND and NAND of literals, has been introduced in the logic array of the high-speed EPLD CY7C361 chip from Cypress. In this paper we give an algorithm for the minimization of SUM-OF-CDEC (SOC) expressions. This algorithm produced the minimum solutions on all small single-output functions, as required by this chip. We propose also the new concept of a CDEC-PLA and show its advantage over the existing AND/OR PLAs. This structure realizes multi-output SUM-OF-CDEC (SOC) expressions, which generalize the well known SUM-OF-PRODUCTS expressions of PLAs. An efficient heuristic algorithm for the minimization of the multi-output SOC expressions is given. We also propose a new approach for solving covering/factorization problems, called Conditional Graph Coloring. This method is used to minimize SOCs, but can be used in other minimization algorithms for circuits with few logic levels. Comparison of several versions of the algorithm on benchmarks proves that our fast multi-output algorithm generates high-quality solutions and is especially efficient on strongly unspecified functions.

1. Introduction.

A Programmable Logic Device (PLD) from Cypress Semiconductor, CY7C361 [7], has been optimized for the realization of very fast asynchronous (self-synchronized) sequential controllers by means of concurrency and partitioning of Finite State Machines (FSMs). This approach leads to Tokenized State Machines (TSM), a new concept which generalizes several known notions of micro-parallelism and was used to synthesize a number of practical controllers [2]. Controller can be designed from multiple deterministic and non-deterministic finite state machines, and mapped into this new device.

The input array of CY7C361 consists of an array of logic gates feeding, in a one-for-one fashion, the state macrocell array. Each logic gate in the array, called a Condition Decoder (CDEC), is the product of a product term, and the complement of a product term (CDEC $= P \cdot \overline{Q}$, where P and Q are products of literals). The inputs to each gate come from any external input to the device and from a number of feedback connections from macrocells, arranged in a segmented array. Therefore, compared to traditional PLA architectures where the excitation functions of the state macrocells are Sum-of-Product (SOP) expressions, the excitation functions of the *CY7C361* macrocells are *Sum of CDEC* (SOC) expressions (where OR gate for summing has a fan-in of 2).

The CDEC gate has been shown to be an excellent choice for a very fast realization of conditions to state macrocells [1, 2]. This fact was confirmed by analysis of structures of many real-life partitioned FSM controllers. This gate is a good fit for the commonly encountered simple branch and join conditions of concurrent FSMs. A problem that is related to the *"device*

fitting problem" [16] of the Cypress CY7C361 chip is the *SOC Minimization.* Since the CY7C361 has only 32 macrocells, a high quality logic minimization to reduce the number of macrocells is very important, and as the real-life examples demonstrated, the SOC minimization can often transform a non-fittable netlist into a fittable one.

The SOC minimization problem was solved as a part of WARP [8, 17, 3, 22], a Cypress Corporation's PLD synthesis system based on behavioral VHDL [3]. WARP encompasses a unique combination of high-level synthesis, logic synthesis, and fitting. In this paper we present a CDEC minimizer that is superior to the one used in [8, 17].

The goal of this paper is, however, more general. We believe that the CDEC gate and related logic minimization methods can find applications both in custom standard cell-based design, and in programmable structures. For instance, CDEC gate can be used as a general-purpose complex gate in layout generators for custom design layout [10]. Another application are new PLD structures, such as the CDEC-PLA proposed below. The Cypress chip is therefore not the sole motivation of our work, and this paper presents the SOC minimization as a **generic logic synthesis problem.** Since these algorithms give good results for very strongly unspecified functions defined by ON and OFF sets, they are applicable in Machine Learning (ML) [18] and Knowledge Discovery from Data Bases (KDD) as well.

In this paper, we introduce an original approach to solve SOC covering problems without generating prime CDEC-implicants. The method is based on a new type of graph coloring algorithm, similar to the ones from [14, 6, 15]. Instead of solving the covering problem with prime CDEC-implicants, we solve the coloring problem for a graph whose nodes correspond to minterms

or some product implicants of a new type. Graph-coloring problems can be solved using heuristic, non-backtracking algorithms for an approximate minimization, or optimal backtracking algorithms for an exact solution with the minimum number of implicants. In graph theory literature, graph-coloring algorithms are presented that trade-off the quality for speed [11, 12] and are applicable to large graphs.

Tsutomu Sasao has proven [21] that for "nearly all functions" three levels is enough. This important result should stimulate further research in various types of three-level minimizers. Our paper follows this line of research for the case of three-level OR/AND/OR circuits implemented as Sum of CDECs.

The paper is organized as follows: Section II defines the CDEC-PLA, discusses the possible approaches to SOC Minimization (CDEC covering) and shows their difficulties. Section III introduces the notion of conditionally compatible cubes. In section IV a *Conditional Graph Coloring Problem* (CGCP) is introduced. A heuristic solution algorithm for the CGCP is given in section V. This algorithm starts from arbitrary product implicants and is used to minimize single-output Boolean functions in a SOC form. An additional process of splitting the input cubes as a preprocessor to this algorithm, in order to improve the results, is presented in section VI. In section VII the entire approach is extended to multi-output functions. Section VIII discusses experimental results, and section IX concludes the paper.

2. Discussion of CDEC Covering

Definition 1 . *A* **CDEC-implicant** *is an implicant of the form* $P \cdot \overline{Q}$ *where* P *and* Q *are products of literals on input variables.*

Definition 2 . *A* **prime CDEC-implicant** *is a CDEC-implicant that is not totally included in any other CDEC-implicant (i.e, the set of minterms covered by it is not included in any other CDEC-implicant).*

From these definitions, product terms and prime implicants of SOP are special cases of CDEC-implicants, but a prime SOP implicant is not necessarily a prime CDEC, because it can be included in a CDEC-implicant.

A standard PLA is realized from two planes: an input AND plane and an output OR plane. Similarly, we propose here the concept of a *CDEC-PLA*. It is composed of a standard output *OR Plane* and a *CDEC Plane* that replaces the AND plane of standard PLAs. The CDEC plane is composed of CDEC gates, the outputs of which are the inputs to the OR plane. SOC minimization is based on covering minterms with *CDEC-implicants*, which we will call *CDEC-covering*. A CDEC-implicant is an implicant that can be realized with a single CDEC gate. CDEC gate $P \cdot \overline{Q}$ has an *AND-part* P and a *NAND-part* $\overline{Q}$. Sometimes the NAND part will be called also an *OR-part*, because for $P = ab$ and $Q = cd$, $ab\overline{cd} = ab(\bar{c} + \bar{d})$. The concept of a CDEC-implicant is a generalization of a prime implicant (a prime) and a product implicant [4, 14, 6] used in standard SOP minimizers such as Espresso [4, 20].

Definition 3 . *An* **essential prime CDEC-implicant** *is a prime CDEC-implicant that covers a minterm that is not covered by any other prime CDEC-implicant.*

The CDEC minimization process results in a SOC expression. Assuming that CDEC is realized by a single gate (as is the case technologically [3]), the exact SOC representation of function f has never more gates than a SOP of

this function because the set of all CDEC implicants includes the set of all prime implicants. The approach to SOC from WARP [8, 17] is the following: (1) Generate all prime implicants of function f (using Espresso). (2) Factorize the prime implicants to prime CDEC-implicants. (3) Solve the covering problem with true minterms as columns and prime CDEC-implicants as rows, such that the number of CDEC-implicants is minimal. This approach has, however, several drawbacks: (1) Since not all prime CDEC-implicants can be generated by factoring prime implicants, the minimum covering with prime CDEC-implicants (called **CDEC covering** from now) will be lost. (2) The generation of prime CDEC-implicants based on standard factoring procedures is not efficient. (3) Since the number of prime CDEC-implicants is much larger than that of prime implicants, a large covering table is created, which cannot be solved even for those functions that the exact minimal prime covering would be still possible to find. (4) Using minterms in the table limits the size of problems that can be solved.

Notice that in order to generate all prime CDEC-implicants one has to start from some kind of product implicants that are not necessarily prime implicants and are smaller than prime implicants.

The following example will illustrate the above ideas.

Example 1. A function for which the approach based on factoring prime implicants can not find the minimum CDEC cover, is given in Figure 1a. The function f can be represented as the sum of three essential prime implicants: $f = \bar{c}d + \bar{a}b\bar{c} + abd$. Based on these implicants, f can not be further factorized to CDEC-implicants. The CDEC realization would consist of 3 CDEC gates and the CDEC-implicants would be identical to the prime im-

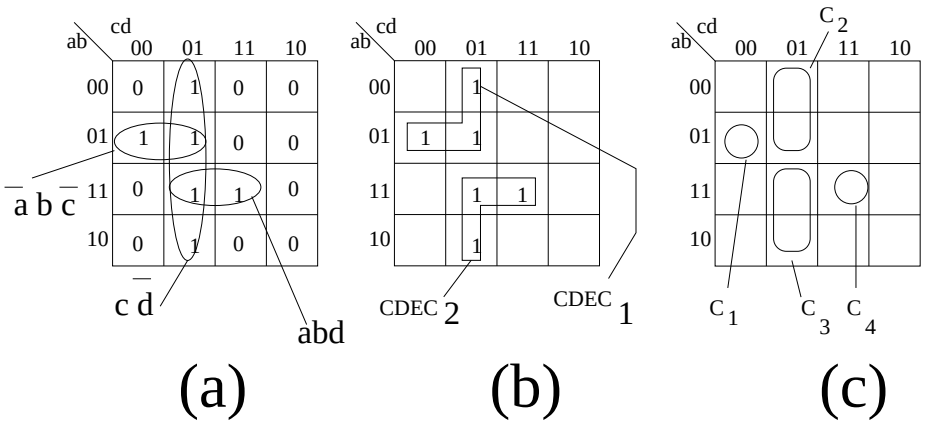


Fig. 1. *(a,b) Function that demonstrates that the minimum SOC cannot be obtained by factorization of primes, (c) Compatible and conditionally compatible cubes from Example 1. Cubes C_2 and C_3 are compatible, cubes C_1 and C_2 are conditionally compatible, cubes C_1 and C_4 are incompatible.*

plicants. However, function f can be realized with only two CDEC gates. The minimal solution can be obtained by the factorization of product implicants that are not primes, as shown in Figure 1b: $f = \bar{a}b\bar{c} + \bar{a}\bar{c}d + a\bar{c}d + abd = \bar{a}\bar{c}(b+d) + ad(\bar{c}+b) = \bar{a}\bar{c}\overline{(\bar{b}\bar{d})} + ad\overline{(c\bar{b})} = CDEC_1 + CDEC_2$.

Two important principles can be learned from this example: (1) Prime CDEC-implicants must be created not only from prime implicants, but also from some product implicants *included* in prime implicants. (2) Essential prime implicants are useful to create essential prime CDEC-implicants, that are included in the exact minimum CDEC cover. Observation (1) can be used in an algorithm to create all prime CDEC-implicants. Observation (2) can be used in both exact and approximate algorithms to find CDEC coverings more efficiently by first finding the essential and secondary essential CDEC-implicants, and thus reducing the combinatorial search, similarly as it is

done in standard SOP minimization algorithms. Our approach is based on graph coloring [15], rather than on covering, but it still generates implicants starting from the essential ones and next the secondary essential implicants. The advantage of coloring is that the large sets of CDEC implicants are never generated to be stored at once in memory.

A basic, exact minimization algorithm for a single-output SOC network may be created similarly to the classical SOP minimization algorithms: (1) Find all prime CDEC-implicants. (2) Find the covering of all true minterms with prime CDEC-implicants, that minimizes the number of CDEC-implicants in the cover (or the total cost of literals in these CDEC-implicants). This kind of exact approach could be used only for functions with a small number of input variables. The number of prime CDEC-implicants increases rapidly with the number of minterms, especially for functions with many don't cares. It is well-known that the set of primes can become too large to enumerate even if it is possible to find the exact minimum cover [5, 4, 20]. A similar property can be shown for prime CDEC-implicants. Moreover, the number of CDEC-implicants is even more dramatically large. Thus, the application of algorithms based on generating *all* primes or *all* prime CDEC-implicants is limited because of results of this kind. It can also be shown that an attempt to reduce the size of the covering problem by removing the primes included in prime CDEC-implicants can lead to a loss of the CDEC cover with minimum literal cost. In addition, both the CDEC-implicant generation and the CDEC-covering problems are NP-hard. More importantly, the covering table can become too large to store in the memory. Therefore, we have chosen to base our approach on graph coloring. With this approach only one NP-hard problem is solved instead of two. For large examples,

the graph occupies much smaller memory than would the respective covering table. This way, when the exact coloring approach is used, the exact minimum term solution is found from the graph. With some modification of backtracking, the minimum literal can be also found. In this paper, the exact graph coloring will be not presented, and only a fast approximate algorithm, which is of more practical value, will be discussed (the exact algorithm is quite similar to the one shown below, just backtracking is added).

The graph for coloring, called *Graph for CDEC-Compatible Coloring* (GCCC), can be created with arbitrary cubes from the ON-set of the function f as nodes. For instance, these can be: minterms, primes, disjoint product implicants [9], minimal product implicants [14, 6], or the (optimal) split cubes introduced below. The advantage of using minterms is the guarantee of the optimum solution if exact coloring for GCCC is found. The disadvantage of minterms is the large size of the graph for completely specified functions with many inputs (if the number of inputs is n, the number of minterms can be of the order of 2^{n-1}, and the graph will be too big to construct). The advantage of using arbitrary cubes instead of minterms is the improved execution speed, but the minimum CDEC cover can be lost (as, when using prime cubes in Example 1). The algorithm presented in section IV can create the graph from **any** type of cubes. We use, however, a special kind of cubes called *split cubes*, for their efficiency and solution quality. For split cubes, the minimal solution can be in theory lost, but experimental results show that this never happened on the tested benchmarks.

3. Conditionally Compatible Cubes

Boolean functions in our system are represented as ON and OFF arrays (sets) of cubes. The following notation will be used: *ON(f)* is the set of ON-cubes of *f*, *OFF(f)* is the set of OFF-cubes, and *DC(f)* is the don't care set. Set DC is implicit in our approach and only ON and OFF are stored. **Cube** C_i is a string of 0's, 1's, and X's; it represents a product of literals of the function *f*. A **product implicant** is an implicant being a cube. A **prime implicant** is a product implicant which is not included in any other prime implicant of that function [4]. Standard notions of literals, sum of products, essential and secondary essential prime implicants [15], consensus, sharp, disjoint sharp, intersection and Hamming distance of cubes will be used. The cubes C_i below can be of any kind, if not mentioned otherwise. The main cube operator used in our approach is the *supercube* [4]. The supercube of two cubes C_i and C_j is denoted by $C_i \uplus C_j$. When the positional cube notation is used, the supercube operator corresponds to the component-wise Boolean OR of the two cubes. This is the smallest cube that includes both C_i and C_j.

Example 2: $C_i = X101$, $C_j = X110$, $C_i \uplus C_j = X1XX$ is the supercube of C_i and C_j.

We say that *two cubes overlap* if their intersection cube is non-empty.

Definition 4 *Two cubes* $C_i \in ON(f)$ *and* $C_j \in ON(f)$ *are* **compatible** *if their supercube* $C_i \uplus C_j$ *does not overlap with any cube* $C_k \in OFF(f)$ *(i. e. does not include a false minterm):*

$$(C_i \uplus C_j) \sqcap OFF(f) = \emptyset \quad \Rightarrow \quad C_i \text{ and } C_j \text{ are compatible}$$

These cubes can be combined to one CDEC-implicant (in this case, prime implicant) with their supercube $SU_{ij} = C_i \uplus C_j$ as the AND-part (the NAND-part is 1).

Definition 5 *Given are two cubes C_i, $C_j \in ON(f)$ and their supercube $SU_{ij} = C_i \uplus C_j$. If the supercube SU_{ij} overlaps with OFF(f), the intersection is called the* **OFF-part** *$OFFP(SU_{ij})$ of the supercube SU_{ij}:*

$$OFFP(SU_{ij}) = OFF \sqcap SU_{ij}.$$

The supercube $\biguplus_{C_l \in OFFP(SU_{ij})} C_l$ of all cubes of the OFF-part $OFFP(SU_{ij})$ is denoted by $SOFFP(SU_{ij})$.

Definition 6 *Two cubes C_i and C_j are called* **conditionally compatible** *if cube $SOFFP(SU_{ij})$ does not intersect C_i nor C_j. Such cubes can be combined to one CDEC-implicant $(SU_{ij} \cdot \overline{SOFFP(SU_{ij})})$, with cube SU_{ij} as the AND-part and expression $\overline{SOFFP(SU_{ij})}$ as the NAND-part. The cube $SOFFP(SU_{ij})$ is called the* **condition cube** *under which C_i and C_j are conditionally compatible.*

Definition 7 *Two cubes are* **incompatible** *if they are not compatible nor conditionally compatible. Such cubes cannot be combined to a single CDEC-implicant.*

Definition 8 *The set Π of cubes is called a* **set of compatible cubes** *if each pair $C_i, C_j \in \Pi$, is either compatible or conditionally compatible with respect to the same condition cube $SOFFP(\biguplus_{C_i \in \Pi} C_i)$. The set Π can be described by a single CDEC-implicant:*

$$(AND\ part, NAND\ part) = (\biguplus_{C_i \in \Pi} C_i\ ,\ \overline{SOFFP(\biguplus_{C_i \in \Pi} C_i)})$$

Example 3. The Boolean function f from Example 1 (Fig. 1) is represented by the cubes: $f = \bar{a}b\bar{c}\bar{d} + \bar{a}\bar{c}d + a\bar{c}d + abcd$. Let $C_1 = \bar{a}b\bar{c}\bar{d}$, $C_2 = \bar{a}\bar{c}d$, $C_3 = a\bar{c}d$, $C_4 = abcd$ (shown in Fig. 1c). The cubes C_2 and C_3 are compatible, because their supercube $C_2 \uplus C_3 = \bar{c}d$ does not intersect the OFF-set of f. The cubes C_1 and C_2 are conditionally compatible under the condition cube SOFFP$(\bar{a}\bar{c}) = \bar{a}\bar{b}\bar{c}\bar{d}$. Their supercube $C_1 \uplus C_2 = \bar{a}\bar{c}$ overlaps the OFF-set of f. Therefore the OFF-part of $C_1 \uplus C_2$ is given by $OFFP(C_1 \uplus C_2) = \bar{a}\bar{b}\bar{c}\bar{d}$. However, SOFFP$(\bar{a}\bar{c})$ does not overlap C_1 or C_2. The cubes C_1 and C_4 are not compatible, because their supercube $C_1 \uplus C_4 = b$ overlaps the OFF-set of f and the supercube of the OFF-part of $C_1 \uplus C_4$ overlaps the cubes C_1 and C_4. Sets of conditionally compatible cubes are: $\{C_1, C_2\}$ and $\{C_3, C_4\}$. The cubes C_1 and C_2 are of Hamming distance-1, i.e. there is exactly one variable d that has different sets of truth values. We call them distance-1 cubes. C_1 and C_4 are not of distance-1, because there are three variables a, c and d that have different sets of truth values, so Hamming distance is equal 3.

4. Conditional Graph Coloring

The input of the *Conditional Graph Coloring* algorithm is the non-ordered graph GCCC = (SMI, RSN, RSC), where SMI is the set of nodes corresponding to (arbitrary) product implicants (in particular, minterms) of function f. RSN is the set of non-directed *normal edges* and RSC is a set of non-directed *conditional edges* between the nodes. The normal edges are drawn as continuous lines, and the conditional edges as dotted lines. Two nodes (i.e., cubes), $MI_i \in SMI$ and $MI_j \in SMI$, are connected by a *normal edge* if MI_i and MI_j are incompatible. Such cubes must be colored with different

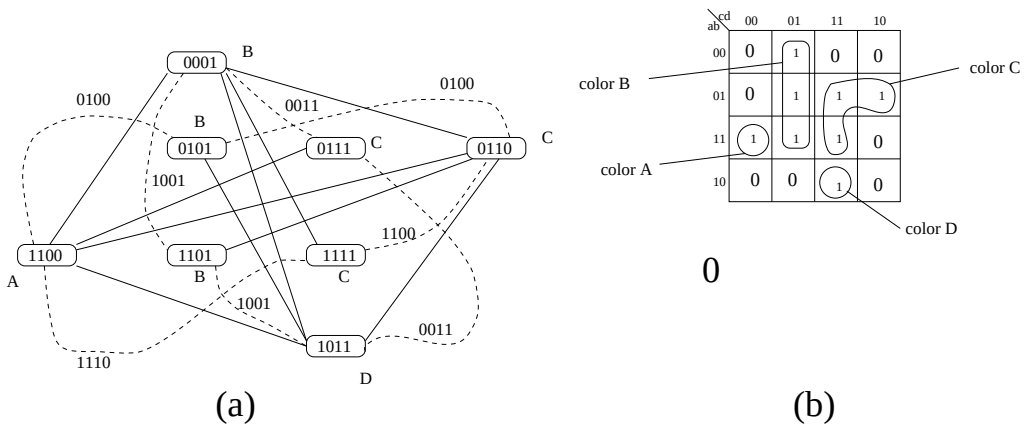


Fig. 2. *Compatible Coloring: (a) graph GCCC for Examples 4 and 6 with a CDEC-minimum compatible coloring shown near nodes, (b) minimum CDEC-implicant cover corresponding to the compatible coloring found in (a).*

colors. If there is no edge between nodes, these nodes can be colored with the same color. MI_i and MI_j are connected by a *conditional edge* if MI_i and MI_j are compatible under the condition indicated by the label $l(SC_{ij})$ of the edge. The label $l(SC_{ij})$ of the conditional edge SC_{ij} connecting the nodes MI_i and MI_j, is calculated as SOFFP(SU_{ij}). If the group of nodes have the same label on all its conditional edges, the nodes are compatible without any additional restrictions to be checked. That means, the labels of the conditional edges are not taken into account by the coloring algorithm, and the nodes can be colored with the same color (what to do when this condition is not satisfied, see Observation 1 after Example 5).

We will illustrate the minimal conditional coloring method on two examples.

Example 4. Fig. 2b illustrates a minimum cover with CDEC-implicants in a

Karnaugh map. These CDEC-implicants were created by CDEC-compatible coloring of the nodes of the GCCC graph from Fig. 2a. Color A describes a CDEC-implicant $ab\overline{c}\overline{d}$. Color B corresponds to a CDEC-implicant $\overline{c}d\overline{a\ \overline{b}}$. Color C corresponds to a CDEC-implicant $bc\overline{a\ \overline{d}}$. Color D corresponds to a CDEC-implicant $a\overline{b}cd$. The nodes of the graph correspond to the minterms in the Karnaugh-Map. The dotted edges labeled with a cube are the conditional edges (with their label given by the cube associated to the edge). The continuous edges are the normal (unconditional) edges. The nodes connected by an unconditional edge must be colored with different colors. A set of nodes, that have the same color, describe a CDEC-implicant. For instance, the nodes given by the minterms 0111, 0110 and 1111 are colored with the same color, C. The supercube of all those nodes is 0111 $\uplus$ 0110 $\uplus$ 1111 = X11X. The only cube in $OFF \sqcap X11X$ is 1110, which does not overlap with the minterms 0111, 0110, or 1111. Thus CDEC-implicant $bc\overline{a\ \overline{d}}$ is generated to cover all nodes colored with color C. The solution found from the minimal conditional coloring, $g = ab\overline{c}\overline{d} + \overline{c}d\overline{a\ \overline{b}} + bc\overline{a\ \overline{d}} + a\overline{b}cd$, has the minimum number of terms, but not necessarily the minimum number of literals. Another coloring could find the minimum literal cost solution as $g' = \overline{a}d\overline{c} + \overline{a}bc + ab\overline{c} + bcd$. Thus, the literal cost should be calculated by the graph-coloring procedure as a secondary cost for all solutions which have the same number of colors. Observe, that the literal minimum solution g' from Example 4 could not be found from a cover with prime CDEC-implicants, since all the CDEC-implicants from this solution (in this case, primes) are included in some prime CDEC-implicants.

Example 5. Using the minimal conditional coloring method to the function

from Fig. 1a a GCCC with nodes C_1, C_2, C_3, and C_4 is created. The coloring of nodes C_1 and C_2 with color A and nodes C_3 and C_4 with color B produces an exact CDEC cover from Fig. 1b, found earlier in Example 1. Note that nodes of the graph are not minterms but some larger product implicants.

In Example 4 all groups of nodes with the same color have only one label on each edge connecting any two nodes from the group. In general, there can be several labels associated with edges between nodes in the group of nodes.

Observation 1. The labels of the set $\{SC_i, \ldots, SC_{i+k}\}$ $(k \geq 2)$ of conditional edges $SC_i \in RSC$ have to be taken into account by the conditional coloring algorithm only if the set of nodes $\{MI_j, \ldots, MI_{j+l}\}$ $(l \geq 3)$ connected by set of edges $\{SC_i, \ldots, SC_{i+k}\}$ is attempted to be colored with the same color. Coloring these nodes with the same color is possible only if the supercube of all the labels in this set, $l(SC_i) \uplus \ldots \uplus l(SC_{i+k})$, does not intersect any nodes from the set $\{MI_j, \ldots, MI_{j+l}\}$.

Definition 9 *A* **compatible coloring** *of each node with a single color is a coloring of nodes such that any two nodes linked by a normal edge have different colors and any set of nodes colored with the same color is a compatible set of nodes. A* **compatible set of nodes** *is a set in which the nodes correspond to a set of compatible cubes defined in Definition 8.*

Definition 10 *The solution to the CDEC-covering problem with the exact minimum number of CDEC-implicants is called the* **CDEC-minimum solution.** *A solution with the minimum number of literals among all CDEC-minimum solutions is called the* **CDEC-minimal-literal solution**.

Definition 11 *The* **literal minimum solution** *is the CDEC-minimum solution which among the solutions with the minimum number of colors selects the solution that has the minimal total cost of literals.*

Definition 12 *The* **minimal compatible graph coloring** *is such coloring that is compatible, minimizes the number of colors, and minimizes the literal cost among all CDEC-minimum solutions.*

Results with a better literal cost can be obtained when a node is allowed to be colored with several colors, which is called *multi-coloring.*

Definition 13 *The* **minimal compatible graph multi-coloring** *is such multi-coloring (which means, any node can be colored with many colors) that is compatible (which means, for any two nodes linked by an edge, their sets of colors are disjoint), minimizes the number of colors, and minimizes the literal cost as the secondary cost function.*

Example 6. Multi-coloring of nodes for a function from Example 4 would create, among others, a solution $g" = \overline{c}d\overline{a\ \overline{b}} + abc\overline{\overline{d}} + cd\overline{a\ \overline{b}} + a\overline{b}\overline{\overline{c}\ \overline{d}}$. The multi-coloring method creates, in general, larger CDEC-implicants, therefore, in contrast to the solutions from Example 4, the above solution has no hazard in ones. (Hazard in ones exists when there are two adjacent implicants, and two true minterms in them that are not covered together by a single third implicant).

The exact minimization of the number of CDEC-implicants is based on the following Theorem.

Theorem 1. If a graph GCCC is created with minterms as nodes and the (multi)coloring of the GCCC is compatible and has the minimum number of

colors (is an exact compatible (multi)coloring), then the CDEC covering created from this coloring has the exact minimum number of CDEC-implicants.
Sketch of a Proof. The number of colors in a compatible coloring is the same as the number of CDEC-implicants. Because nodes of GCCC correspond to minterms, all possible CDEC-implicants are created in all potential colorings. (The complete formal proof is similar to one from [6]).

Our algorithm to solve the compatible coloring problem is called conditional graph coloring algorithm. In the corresponding program, called SOCMIN, we first use Algorithm 2 from section 6 for optimal splitting of initial cubes to smaller cubes for which optimal compatible coloring could be found more efficiently (we call this new type of cubes the *"split cubes"*). For multi-output functions the known *Muller transform* is used to convert a multi-output function to a single-output function by introducing new input variables whose binary combinations encode the output functions. Because only some combinations of new variables are used, the new function is very strongly unspecified (has very many don't cares). We found this approach very efficient for data represented with ON, OFF covers, which contradicts the previous opinion that Muller method is impractical. We believe these opinions resulted only from using ON, DC covers for representation, which caused the blow-up of data. In our representation, the algorithm with Muller method as a preprocessor is very fast. Observe, that this approach can be used to **extend arbitrary single-output minimizer that uses ON,OFF representation for (in)complete functions into a multi-output minimizer**. Section 5 will present our fast conditional coloring algorithm in more detail.

The design objectives of this algorithm were: **(1)** size of problems that can be handled, **(2)** speed, **(3)** ability to find optimal solutions for non-cyclic functions. Although it works with any functions, its advantages are clearly seen for strongly unspecified and non-cyclic functions. Interestingly, a characteristic of our approach is that *"the more unspecified is the function, the better are our results, comparing to the known tools".*

5. Realization of the Conditional Graph Coloring algorithm.

The basic idea of our approximate GCCC coloring algorithm is to color successively, the most restricted nodes with the most adjacent *normal edges.* By that, colors are chosen, that result in the smallest number of restrictions for all other nodes.

A minterm from an essential implicant that is covered by only this implicant has the most adjacent edges in the graph, so it will be colored first by our algorithm. Although this is a graph-coloring algorithm, its operation is in essence similar to the well-known SOP-minimizing algorithm that extracts the essential and next (iteratively) the secondary essential primes, thus being able to find the exact solution (without setting the covering table) for non-cyclic functions. Therefore, our graph coloring algorithm gives very good results on non-cyclic functions. It is known that non-cyclic functions are found quite frequently in real-life applications, [4].

Definition 14 *A* **Color-Restriction** *of node MI_i is a labeling of MI_i with the complement of the color that MI_i is not allowed to be colored with.*

Definition 15 *A node MI_i of the Graph GCCC is associated with the* **Color-Restriction Vector** *that includes all the restrictions of node MI_i. The*

color restrictions result from the coloring of other nodes of the Graph and are the colors that node MI_i is not allowed to be colored with.

Definition 16 *A* **k-Color-Restriction** *of a node MI_i is the labeling of MI_i with the Color-Restriction Vector having k elements.*

Figure 3 shows an example of a node MI_i with a 3-color-Restriction. Node MI_1 cannot be colored with colors A, C, and G. When node MI_i is colored in the algorithm, the Color-Restriction Vector is determined or updated for every adjacent node MI_j, $i \neq j$. Based on these Color-Restriction Vectors and the Resulting Color-Restrictions of coloring for the candidate nodes, the node MI_i is determined to be colored next, and the color c_i of this node is determined. A Resulting Color-Restriction is a restriction obtained for a possible coloring of MI_i with the candidate color c_i.

Definition 17 *A* **Candidate Node** *MI_i is a (non-colored) node of the graph GCCC that is a candidate for the next node to be colored.*

The evaluation whether MI_i is a good candidate for the next node to be colored is based on the following ordered criteria:

(1) the maximal k-Color-Restriction of the candidate,

(2) the degree of normal edges,

(3) the numerical parameters from the Resulting Color-Restriction Matrix,

(4) the degree of conditional edges.

Definition 18 *A* **Candidate Color** *is a color c_i that the Candidate Node MI_i can be colored with. Thus, c_i is not included in the Color-Restriction Vector of MI_i.*

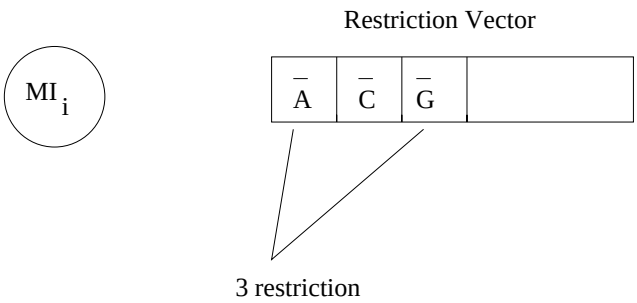


Fig. 3. *Example of a node MI_i with a 3-color-Restriction. Node MI_i cannot be colored with colors A, C, or G.*

Definition 19 *The* **Resulting Color-Restriction Matrix** *of a node MI_i is a matrix that, as columns, includes all the k-Color-Restrictions of nodes MI_j, $i \neq j$ resulting from a coloring of MI_i $i \neq j$ with color c_i representing the rows of the matrix.*

If the index of the column is denoted with k and the index of the row is denoted with c, an element a_{ck} of the Resulting Color-Restriction Matrix indicates how many k-Color-Restrictions result from coloring of node MI_i $i \neq j$ with color c.

Example 7. Figure jour4 shows an example of a Resulting Color-Restriction Matrix for the node MI_1 (denoted by 1 in Fig. 4a). Assume that node MI_1 can be colored with A, B or C. The edges in Fig. 4 are *normal edges*, and the Color-Restriction Vectors of the nodes 2 through 6 are shown besides the nodes. If node 1 is colored with A, node 2 becomes $\overline{A}\ \overline{D}$ (2-Color-Restriction), node 3 becomes $\overline{A}\ \overline{C}\ \overline{D}$ (3-color-Restriction), node 4 becomes $\overline{A}\ \overline{B}\ \overline{D}$ (3-color-Restriction), node 5 becomes $\overline{A}\ \overline{B}\ \overline{C}\ \overline{D}$ (4-color-Restriction), and node 6 becomes $\overline{A}\ \overline{C}$ (2-color-Restriction). Thus, the new labeling would have one node with a 4-color-Restriction, two nodes with a 3-color-

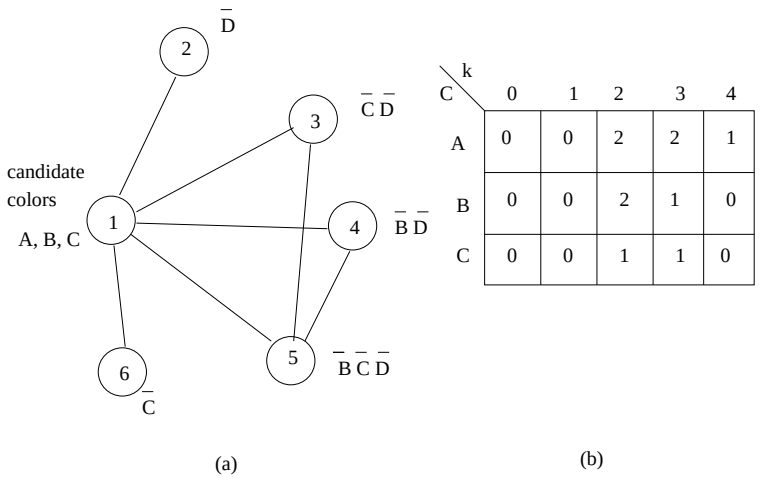


Fig. 4. *Color Restriction Matrix: (a) candidates for coloring, (b) resulting Color-Restriction Matrix for Candidate node* MI_1

Restriction and two nodes with a 2-Color-Restriction (as in the first row of the matrix from Fig. 4b). Similarly, if node 1 were colored with C, node 2 would become $\overline{C}\ \overline{D}$, and node 4 would become $\overline{B}\ \overline{C}\ \overline{D}$ (as in the second row of the matrix). From Fig. 4, the best coloring of node MI_1 is color C, because the resulting restriction is minimal. The coloring of MI_1 with A results in a 4-color-Restriction whereas the highest restriction for the coloring with B or C are 3-color-Restrictions. C has in the matrix a 2-color-Restriction of 1 while B has a 2-color-Restriction of 2, so C is the overal best choice for coloring of node 1.

Definition 20 *The maximal k-Color-Restriction of all colors c in the Resulting Color Restriction Matrix of a node* MI_i *is called the* **Max-Resulting-k-Restriction**.

In our conditional graph coloring algorithm, the selection of nodes to be colored next is done according to the minimal Max-Resulting-k-Restriction.

This means, that this is the first criterion to choose the next node and its color. If there is an equality in the Max-Resulting-k-Restriction, then all the other k-Color-Restrictions of the Resulting Color Restriction Matrix are taken into account as shown in the previous example.

Now we will give the complete Conditional Graph Coloring Algorithm, using standard fonts for commenting on details of the algorithm.

Algorithm 1 for Conditional Graph Coloring.

BEGIN

step 1:

- *select the first node to be colored with color 1 by taking a node with the highest degree of normal edges;*
- *if (more nodes with same degree) then take the node with the highest degree of conditional edges;*

step 2:

- *color the selected node MI_s with the selected color c_s;*
- *make the entries in the Color Restriction Vectors of all nodes MI_i, $i \neq s$;*
- *determine the restrictions resulting from normal edges;*
- *determine the restrictions resulting from conditional edges;*

In order to determine the restrictions resulting from conditional edges, a graph search for all nodes that are connected to MI_s and have the same

color c_s, has to be performed. The result of this search is a connected subgraph $GCCC_S = (SMI_S, RSC_S)$ of the initial graph GCCC. The subgraph has only conditional edges and has the property that its set of nodes (SMI_S) is a set of CDEC-compatible pairs. In order to find the restrictions on the nodes MI_i caused by the conditional edges, every single node MI_{con} that is connected via conditional edges to the subgraph $GCCC_S$, has to be determined. For each of these nodes, MI_{con}, the set $S = SMI_S \cup MI_{con}$ has to be checked if it is a set of CDEC-compatible pairs. Therefore, the supercube of the labels of the conditional edges from RSC_{con} and the conditional edges connecting MI_{con} with the subgraph $GCCC_S$ have to be calculated. If the supercube intersects the cubes assigned to the nodes of SMI_S or the cube assigned to MI_{con}, the set S is not a set of CDEC-compatible pairs, and the color c_S has to be included in the Color Restriction Vector for node MI_{con}. Additionally to the graph search above, all nodes MI_j connected to MI_S by conditional edges have to be explored, if MI_j is conditionally connected to other nodes colored with c_S. If this is true, then a second graph search has to be performed for these nodes, and the resulting subgraph has to be checked for CDEC compatible pairs when MI_j is added to the subgraph.

step 3:

- *remove the normal edges of the node MI_S from the graph GCCC;*

step 4:

- *select the Candidate node MI_T to be colored with the Candidate color c_T;*
- *select according to the following criteria:*

1. *select the nodes with the highest k-Color-Restriction;*

   if (more than one node for criterion 1) then

2. *select nodes with the highest degree of normal edges to the not-colored nodes;*

   if (more than one node for criterion 2) then

3. *select the nodes and colors with the minimal resulting restrictions;*

   if (more than one node for criterion 3) then

4. *select the nodes with the highest degree of conditional edges to the not colored nodes;*

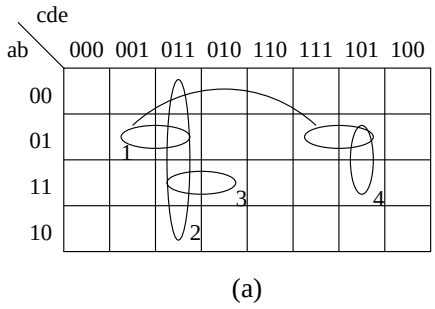
   if (more than one node for criterion 4) then

5. *select the smallest node number and the smallest color;*

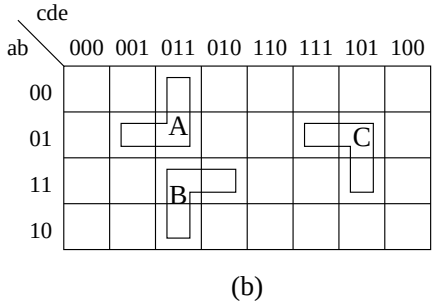
Criterion 3 selects the candidate node according to the Resulting Color Restriction Matrix introduced earlier. Node MI_T and the corresponding color c_T are selected such that the resulting restriction on other nodes is minimal. Therefore, the node and the color with the minimal Max-Resulting-k-Restriction are selected. If there is more than one node with this Max-Resulting-k-Restriction, the next minimal Resulting-k-Restriction is determined, as shown in the example from Fig. 4. In order to determine the resulting restrictions from conditional edges the method described at the end of **step 2** is applied.

step 5:

- *assign the candidate node MI_T and the candidate color c_T from* **step 4** *to the selected node MI_S with the selected color c_S of* **step 2**;
- *if (not all nodes of GCCC are colored) goto* **step 2**;



(a)



(b)

Fig. 5. *Splitting primes to Example 8*

END

6. Splitting of Cubes

The conditional graph coloring gives minimum solution using minterms as nodes and can give good solutions with arbitrary cubes, but, as shown in Example 1, primes should be split in order to create CDEC-implicants for the exact minimum solution. The question is, how to generate "good" (but larger than minterms) initial cubes for coloring. Some essential primes can be used to direct the split, but in general other primes may be useless in this respect, as shown in Example 8.

Example 8. The function from Fig. 5 has four primes, given by $f = \bar{a}be + \bar{c}de + ab\bar{c}d + bc\bar{d}e$. The cubes $\bar{a}be$, $\bar{c}de$, $ab\bar{c}d$, $bc\bar{d}e$ are represented in Figure 5a by cubes 1, 2, 3 and 4 respectively. None of them can be factored to CDEC-

implicants. If prime implicants are used to generate prime CDEC-implicants, the solution, with four prime CDEC-implicants (also prime implicants in this case), is generated. However, as shown in Figure 5b, the same function is realized with only three CDEC-implicants A, B and C. One of the CDEC-implicants; A = $\bar{a}\bar{c}e(b+d)$, is created by factorizing the product implicants $\bar{a}b\bar{c}e$ and $\bar{a}\bar{c}de$ which are not prime implicants. Two other prime CDEC-implicants are created by factoring one prime implicant with one non-prime product implicant. For instance, the CDEC-implicant C = $bce(\bar{a}+\bar{d})$ is created from prime implicant 4, $bc\bar{d}e$, and product implicant $\bar{a}bce$, included in prime implicant 1, $\bar{a}be$. Observe, that essential prime implicants 3, $ab\bar{c}d$, and 4, $bc\bar{d}e$, are used to create prime CDEC-implicants B = $a\bar{c}d(b+e)$ and C = $bce(\bar{a}+\bar{d})$, respectively. Thus, the solution with three CDEC-implicants is: $f = \bar{a}\bar{c}e(b+d) + a\bar{c}d(b+e) + bce(\bar{a}+\bar{d})$.

Examples 1 and 8 suggest that cube-splitting operations have to be performed on the input functions before invoking the graph coloring routine, and that essential and secondary essential prime implicants and CDEC-implicants play a role in splitting. A heuristic algorithm splits the given input cubes and generates a *set of split cubes.* This new set of cubes, when used as nodes in graph GCCC for the conditional graph coloring algorithm, gave exactly the same cost solutions as the solutions found from the graph created from minterms **in all tested functions**. This means that the quality of our splitting algorithm presented below is very high, but we have no proof of its optimality.

Definition 21 *Distance-1 graph of cubes is a graph with nodes corresponding to cubes and edges corresponding to pairs of cubes that have Hamming*

distance not greater than 1.

Example of such graph for cubes from Figure 7a is shown in Figure 7b.

Definition 22 The set r-d1-set *is a set of nodes, which are distance-1 from the candidate node in the distance-1 graph, and are not compatible or conditionally compatible with the candidate node in GCCC.*

A new cube operation, **Initial-Split**, is defined, which is needed for the cube splitting algorithm.

Definition 23 *Initial-split operation splits the candidate node P with respect to r cubes in the* r-d1-set.

- *cube* $P = (p^1,\ p^2, \ldots,\ p^n)$, *and set* $N = \{1, 2, \ldots, n\}$ *where* p^i, $i \in N$ *are the literals of* P.
- r-d1-set $= \{Q_1,\ Q_2, \ldots,\ Q_r\}$, *and set* $R = \{1, 2, \ldots, r\}$ *where* Q_j, $j \in R$ *are the distance-1 nodes to the candidate node.*
- $Q_j = \{q_j^1,\ q_j^2, \ldots,\ q_j^n\}$, *where* $j \in R$ *where* q_j^i, $i \in N$, $j \in R$ *is the* i_{th} *literal of the* j_{th} *cube in* r-d1-set.
- *Let, for all* Q_j, $j \in R$ *in r-d1-set, set* $Lq^i = \{q_1^i,\ q_2^i, \ldots,\ q_r^i\}$, *where* $i \in N$ *represents the* i_{th} *literal of all* Q_j, $j \in R$.

The **Initial-split cube operation** *is defined as follows: for those* p^i, $i \in N$ *of cube* P, *that have a don't care* X, *if the set* Lq^i *has different truth values, then the don't care* X *in* p^i, $i \in N$ *of cube* P *is split into 0 and 1.*

Example 9. This example illustrates the application of the Initial-Split operation. Given is function $f = \bar{a}b\bar{c}\bar{d} + \bar{c}d + abcd$. The cubes $\bar{a}b\bar{c}\bar{d}$, $\bar{c}d$, and

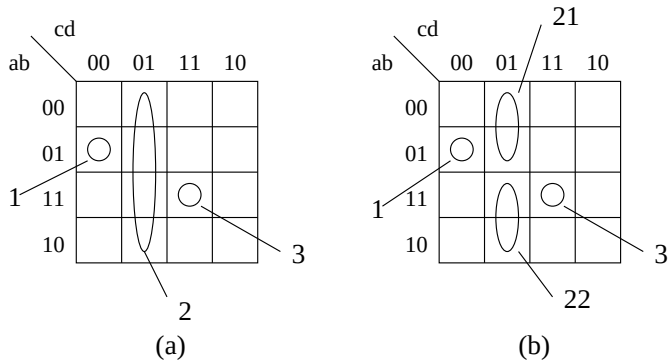


Fig. 6. *Illustration to Example 9 for Initial-split*

$abcd$ are represented by cubes 1, 2 and 3 respectively, in Figure 6a. Cubes 1 and 3 are r-d1-set, cube 2 is candidate node for splitting. We split cube $2 = P = \bar{c}\ d$ with respect to cubes $1 = \bar{a}b\bar{c}\bar{d}$ and $3 = abcd$ and using the Initial-split cube operation. Cube P can be split in two ways with respect to cube 1 (with respect to variable a, and with respect to b) and in two ways with respect to cube 3 (for variables a and b). Only one of these ways, shown in Fig. 6b and created according to Initial-split definition, is good for both cubes 1 and 3 in the sense that the three prime implicants 1, 2, 3 could be replaced with two prime CDEC-implicants, as it was already shown in Fig. 1b. This can be found from cubes' literals, since out of variables a and b that are absent in P (have symbol X in cube P) the variable a is different in cubes 1 and 3, while variable b is the same. Thus, cube $\bar{c}d$ is split with respect to variable a creating cubes $21 = \bar{a}\bar{c}d$ and $22 = a\bar{c}d$.

Definition 24 Initial-split-set of a node *is a set of cubes generated as a result of splitting a node using the Initial-split cube operation.*

Definition 25 *The* **Consensus Set** *is formed by the consensus of each of*

the cubes in Initial-split set and r-d1-set.

The **New-list** consists of the set of cubes generated as a result of splitting a node. The input nodes of the graph GCCC form the **original-list**, which gets updated with split cubes, whenever a node is split. The Algorithm 2 uses two graphs: GCCC and Distance-1 graph.

Algorithm 2, creating Split Cubes from Arbitrary Cubes.

BEGIN

step 1:

If the input cubes are non-disjoint,

> *For all the cubes that are non-disjoint, sharp the cube which has lower degree of distance-1 edges from the cube that has a higher degree of distance-1 edges, until there are no more non-disjoint cubes.*

step 2:

A. *Select the node which has the highest degree of distance-1 edges in GCCC.*

- *If the selected node is a minterm, then choose another node with the next highest number of distance-1 edges in the Distance-1 graph.*
- *If all the distance-1 nodes of the selected node have compatible edges and/or conditional edges in GCCC, then discard that selected node and choose another node with the next highest number of distance-1 edges in the Distance-1 graph.*

B. *If there is more than one node, then choose the node with the highest degree of incompatible edges in GCCC.*

C. *If there is still more than one node, then choose the node with the lowest degree of compatible edges in GCCC.*

step 3:

The candidate node is split with respect to the Distance-1 nodes connected by distance-1 edges in GCCC. Disregard the Distance-1 nodes connected to the candidate node by compatible or conditionally compatible edges. Form the r-d1-set.

step 4:

A. *If the r-d1-set consists of two or less cubes, perform the Initial-split cube operation on the candidate node with respect to the cubes in the r-d1-set and form the Initial-split-set. Go to step 6.*

B. *If the r-d1-set consists of more than two cubes:*

- *Perform the Initial-split cube operation on the candidate with respect to the cubes in the r-d1-set and form the Initial-split-set.*
- *If all the cubes in Initial-split-set are minterms, then go to step 6.*

step 5:

A. *Find the consensus for each of the cubes in the Initial-split-set and each of the cubes in the r-d1-set and form the consensus-set.*

B. *For each cube in the Initial-split-set:*

- *Sharp each cube in the consensus-set with the selected cube in the Initial-split-set and add the resultant cubes to the New-list.*
- *Find the intersection of each cube in the consensus-set with the selected cube in the Initial-split-set and add the resultant cubes to the New-list.*

C. *Remove the repeated cubes in the New-list and add the New-list to the Original-list.*

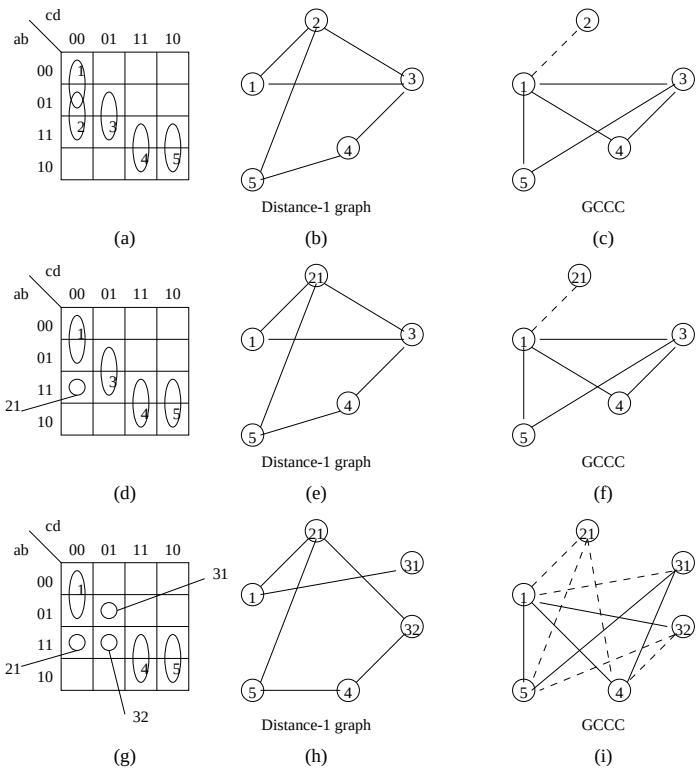


Fig. 7. *Illustration of Algorithm 2 for function from Example 10: (a) initial cubes, (b) initial distance-1 graph, (c) initial GCCC, (d) result of sharping, (g) result of Initial-split, (h),(i) - the state of the graphs when Algorithm 2 terminates. Condition from step 6 is satisfied.*

step 6: *Repeat steps 2 - 5 until there are no more non-split nodes with distance-1 edges that are not covered by compatible or conditional edges.*

END

Example 10. The input cubes are given by the function: $f = \bar{a}\bar{c}d + b\bar{c}\bar{d} + b\bar{c}d + acd + ac\bar{d}$ shown in Figure 7a. The cubes $\bar{a}\bar{c}d$, $b\bar{c}\bar{d}$, $b\bar{c}d$, acd, $ac\bar{d}$ are denoted by 1, 2, 3, 4 and 5, respectively. The distance-1 graph and the GCCC for function f are shown in Figure 7b and 7c. Because the input cube set consists of non-disjoint cubes, step 1 of Algorithm 2 is executed.

Cubes 1 and 2 are non-disjoint and cube 2 has a higher degree of distance-1 edges (3 distance-1 edges) in the distance-1 graph than cube 1 (2 distance-1 edges). Therefore, cube 1 is sharped from cube 2 giving the result $cube21$: $cube1 \ \# \ cube2 = 0X00 \ \# \ X100 = 1100$. Figure 7d shows the input cube list, where all cubes are disjoint cubes. The respective distance-1 graph and the GCCC graph are shown in Figure 7e and 7f. Nodes 21 and 3 have the highest degree of distance-1 edges (3 distance-1 edges) in the distance-1 graph. Because node 21 is a minterm, it is discarded and node 3 is chosen as the candidate node. Now node 3 is connected to nodes 1, 2 and 4 by distance-1 edges. Node 2 is connected to node 3 by a compatible edge, thus, node 2 is discarded and nodes 1 and 4 form the r-d1-set. Hence the candidate node 3 has to be split with respect to nodes 1 and 4 from the r-d1-set. Performing the Initial-split cube operation, gives cubes 31 and 32. Observe from Fig. 7i that nodes 21, 32, 4 and 5 can be colored with color A, and nodes 1 and 31 with color B, which leads to a solution with 2 CDEC-implicants, $y \ = \ a(b+c) \ + \ \bar{a}\bar{c}(b+\bar{d})$.

7. Multiple-Output to Single-Output Transformation

The CDEC minimization based on Graph Coloring, as described, is defined for a single-output Boolean function f_i, specified by the ON-set $ON(f_i)$ and the OFF-set $OFF(f_i)$. To use it for multiple-output functions, therefore, a multi-output to single-output transformation is applied. R.E. Miller presented the so-called *Muller transformation* (attributed to Muller) in [13] that transforms a multi-output function with m input and n output variables to a single-output function with $m+n$ input variables. The method is not practically applicable for multi-output functions that are represented by the ON

and DC sets (such as in Espresso), because it creates as many as $m \cdot (2^n - n)$ additional DC-minterms. This disadvantage is not relevant in our CDEC minimization, because the DC-set is absent from our representation as we store only ON and OFF sets.

Let us illustrate Muller transformation on an example. The set of input variables is given by $\{i_1, i_2, \ldots, i_m\}$ and the multi-output function F is represented by the set $F = \{f_1, f_2, \ldots, f_n\}$, where f_i is a single-output Boolean function.

Example 11. An example of the Muller Transform applied on the set $F = \{f_1, f_2\}$ ($m = 2$ outputs) with $n = 3$ input variables is shown in Figure 8. The transformation results in a single-output function $\tilde{F}$ with $m + n = 5$ input variables. The rows of the Karnaugh map are the m input variables of F, and the columns are the additional input variables of the function $\tilde{F}$ resulting from the n outputs of the set of functions F. The columns of the Karnaugh Map that are *"one-hot encoded"* (columns 01 and 10) represent the single-output functions $f_i \in F$ (f_1 and f_2), and the other columns are filled with DCs. "One-hot encoded" means, that only one literal o_i of the code of the column is "0", all the other literals are "1". If the i-th literal of a one-hot encoded column is "0", then this column corresponds to the function f_i of the set F. The Graph-Coloring-based CDEC minimization generates the CDEC-implicants that are shown in Figure 8. They cover the function $\tilde{F}$. The expression for single-output function $\tilde{F}$ obtained by CDEC minimization is: $\tilde{F} = \bar{i_1}\bar{i_2}\bar{i_3} + i_2\overline{(i_1 i_3)}\ \bar{o_1}o_2 + i_1\overline{(i_1 i_3)}\ o_1\bar{o_2}$. For creation of the expressions of single-output component functions of multi-output function F, respective Boolean constants are substituted for o_1 and o_2, so that the component functions f_1 and f_2 will depend only on the initial

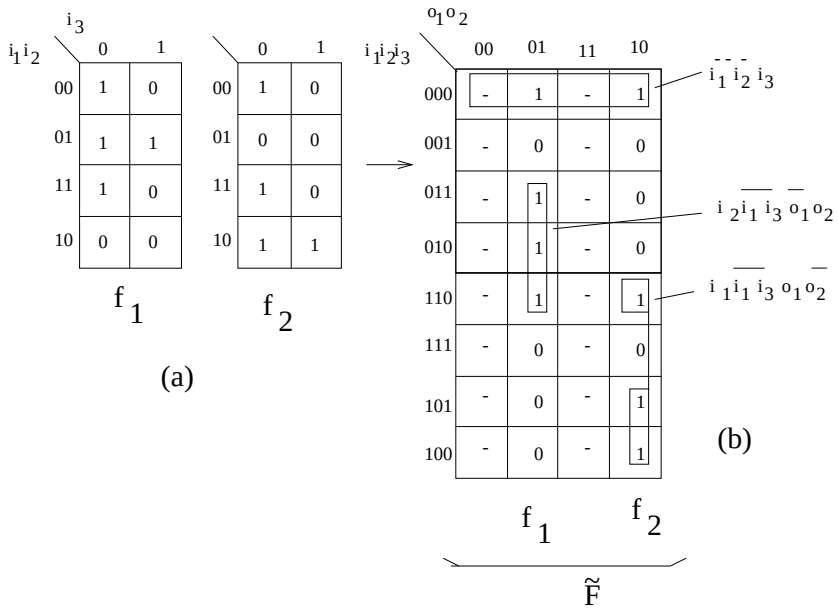


Fig. 8. *Example 11 of the Muller Transform.*

input variables i_1, i_2, i_3. This way, single-output function $\tilde{F}$ is transformed back to a multi-output function F. The CDEC-implicants are $\bar{i_1}\bar{i_2}\bar{i_3}$, $i_2\overline{(i_1 i_3)}$, and $i_1\overline{(i_2 i_3)}$. The additional variables o_1 and o_2 are not included as literals in the CDEC-implicants, they only describe to which single output functions f_i a given CDEC-implicant belongs. The first CDEC-implicant, $\bar{i_1}\bar{i_2}\bar{i_3}$, exists for both substitutions $o_1\bar{o_2}$=1 and $\bar{o_1}o_2$=1 in the formula for $\tilde{F}$. Therefore, it has to be used for both functions f_1 and f_2. Implicant $i_2\overline{(i_1 i_3)}$, exists only for substitution $\bar{o_1}o_2$=1 so it belongs only to function f_1. Implicant $i_1\overline{(i_1 i_3)}$, exists only for substitution $o_1\bar{o_2}$=1 so it belongs only to function f_2. Concluding, three CDEC gates are necessary to realize functions f_1 and f_2:

$$f_1 = \bar{i_1}\bar{i_2}\bar{i_3} + i_2\overline{(i_1 i_3)}, \qquad f_2 = \bar{i_1}\bar{i_2}\bar{i_3} + i_1\overline{(i_2 i_3)}$$

In case of a three-output function (f_1, f_2, f_3), an implicant belonging to function f_1 alone would have subcubes X11, 0X1, 01X, 011, or 0XX on

variables o_1, o_2, o_3. Implicant belonging to functions f_1 and f_2 would have subcube XX1, implicant belonging to functions f_1 and f_3 would have X1X, and implicant belonging to functions f_1, f_2, and f_3 would have XXX. Similarly, a solution with the minimum number of CDEC-implicants can be found for a multi-output function of many outputs, but it is not necessarily the best solution literalwise.

8. Experimental Results

We created a tool, called SOCMIN, and tested it extensively on MCNC, Cypress, and Machine Learning benchmarks. Let us observe that contrary to MCNC and Cypress benchmarks, the Machine Learning benchmarks from U.C. Irvine have a very large number of don't cares (we binary-encoded the multiple-valued variables). We verified all our results with SIS verifier. Some results for multi-output and single-output MCNC benchmark functions are presented in Tables 1, 2, 3 and 4.

The goals of our experiments with SOCMIN were to answer the following questions:

1. How much improvement in the number of implicants or literals is gained by using the CDEC-implicants instead of prime implicants.
2. How good is our heuristic minimization strategy based on cube splitting compared to the exact solutions.

Table 1 has columns for input and output variables, and experiments with various types of initial cubes Start from: Minterms, Espresso, Spectra and SOCMIN with CSA (use *Cube Split Algorithm 2* for preprocessing). Column *Start from Minterms* has two subcolumns; *mntrms*, which includes

the number of minterms, and *CDEC*, which includes the number of CDEC-implicants in exact solutions generated from minterms by SOCMIN. Column for *Espresso* has two subcolumns; *cubes* and *CDEC*. The first subcolumn shows the number of prime implicants in the Espresso solution, and the second column shows the number of CDEC-implicants in the approximate CDEC cover generated by SOCMIN from primes of Espresso as the starting point for coloring. Analogously, column for *Spectra* has *cubes* generated as disjoint cubes obtained from the disjoint cube producing program Spectra [9]. Finally, column "SOCMIN with CSA" shows the numbers of implicants in CDEC cover found by SOCMIN in a GCCC created from split cubes produced by Algorithm 2.

The last column, Time, is given for this experiment (time includes the splitting time for cubes), proving that even with preprocessing our algorithm is very fast. In all cases (which could be compared thanks to the non-excessive number of minterms), the results of Socmin generated using the preprocessing splitting algorithm, have the same number of CDEC-implicants as in the exact minimum generated with the minterms. This demonstrates the that if we start from minterms we can use our algorithm alone, or to precede it with Espresso and splitting to decrease the size of the graph. The program can start from (ON and OFF) minterms or arbitrary cubes. If the data are formulated with cubes, the program splits these cubes optimally. Thus, for large examples the program should use Espresso and cube splitting for preprocessing to allow graph creation.

Next we checked how much we gain in the number of terms compared to Espresso. Some functions, *rd73, rd84, vg2,* demonstrate that the CDEC cover is substantially smaller than the SOP cover - in the case of *rd84* there

are only 30 CDEC-implicants in the cover found by SOCMIN versus 255 primes in the cover from Espresso. Concluding; for multi-output functions, CDEC-covers are better than prime implicant covers, SOCMIN produces "high quality" CDEC-covers, and is very fast (the Times were calculated for the last column).

Table 2 compares the number of CDEC-implicants in solutions of the algorithm for single output MCNC examples for variants starting from minterms, Espresso, Spectra, and *SOCMIN CSA* cube-splitting. We found that on single-output functions, the results of the variant using cubes generated by Espresso as the starting point is usually only slightly worse than using Algorithm 2 (although there are exceptions, like *rd73*3 where SOCMIN starting from splitted cubes generated a cover with only 13 terms, while when starting from Espresso primes the best CDEC cover had as many as 25 terms). It has to be pointed out, however, that in EPLD environment, which was the original motivation of our research, even one gate counts, because it could mean the entire chip, so Algorithm 2 should be always used.

In order to verify its quality on strongly unspecified functions, SOCMIN was tested for ML examples with high percentage of don't cares. As the algorithms used in SOCMIN have been designed for strongly unspecified functions, very good results compared to Espresso have been obtained (Tables 3, 4). The numbers of terms, literals, and DFC were compared (DFC is a standard metric used in ML). DFC was calculated by adding the cardinalities of each of the subfunctions in the decomposition. For arbitrary non-decomposable block in Curtis Decomposition the DFC of the block is calculated as 2^k where k is the number of inputs to the block. In "gate-based" minimizers such as Espresso it then fair to assume that a DFC of

a decomposable gate (such as AND, OR or EXOR) is equal to the total DFC of a circuit equivalent to this gate, that is constructed from two-input gates. The DFC of a four-input AND gate, OR gate or EXOR gate is then $2^2 + 2^2 + 2^2 = 12$, since such gates can be decomposed to balanced trees of three two-input gates. The DFC of a CDEC functor is calculated in the same way, by decomposing it to two-input AND and NAND gates. The number of terms and literals for the CDEC implicants generated using SOCMIN is never greater than the number of terms and literals for the prime implicants generated by Espresso. The DFC is lower in most cases.

In another experiment, to analyze behavior of SOCMIN on strongly unspecified functions, subsets of cares have been removed, and comparisons with Espresso performed. Table 4 presents these results with 25%, 50% and 90% of data replaced with don't cares. As the percent of don't cares increases, SOCMIN gives solutions that are better and better than those of Espresso. There is a huge decrease in the number of terms and DFC, compared to Espresso. This proves that our method works better for very strongly unspecified functions.

Concluding, both the single- and multi-output examples demonstrate that our cube-splitting heuristic works very well on large, and also incompletely specified, functions, and that the CDEC covering has less terms than SOP covers. Another advantage of SOCMIN (of not much importance for CY7C361) is that it works especially efficiently and effectively when compared to Espresso for *very strongly* unspecified functions. Finally, in multi-coloring mode the nodes can be colored with more than one color, which further decreases the number of literals, which can have applications in standard cell and complex gate realizations of SOC.

9. Conclusions

The main contributions of this paper are the following.

1. In order to achieve superior speed in sequencer designs over competing PLD devices, Cypress brought to market an innovative architecture, CY7C361. This architecture introduced a new kind of universal logic gate, the CDEC. Because there is only 32 macrocells in the chip, saving only one CDEC gate can be quite important (the well-known "fit/no-fit problem"). The functions in CY7C361 are relatively small and single-output, which allows for a very high quality algorithm that we have developed. Although we cannot clain that our algorithm is exact, **we did not find any single case that the solution was not the exactly minimum one.**

2. The new concept of CDEC PLA has been introduced in this paper and its practical usefulness to reduce the number of terms has been demonstrated on examples.

3. The respective multi-output algorithms have been developed and compared to known PLA tools. Several variants of such high quality algorithms for a general problem of multi-output CDEC minimization have been implemented. They can find applications in CDEC PLA minimization and as a first step in multi-level factorization for standard cell synthesis. (We found that circuits generated by our program are a good starting point to multi-level factorizing programs.)

4. The algorithm gives very good solutions, better than Espresso, for multi-output strongly unspecified functions, which has several applications,

including controller design, Machine Learning and KDD. Such strongly unspecified functions exist in several areas of logic synthesis: design of large controllers, functions created in multi-level or multi-output optimization problems (such as Muller transform), and also in many areas of Machine Learning and Knowledge Discovery from Databases.

5. We proposed a new general approach to solve covering problems. The approach to solving covering problems introduced here, called Conditional Graph Coloring, can be used in other similar problems such as PLA minimization, or Column Minimization Problem in Curtis-like decomposition of Boolean and Multi-valued Relations [19].

6. The usefulness of Muller transform has been demonstrated. It is very efficient for data representation by ON, OFF covers, and the previous opinion that it is impractical resulted from using ON, DC covers for representation of data, which caused the blow-up of data. This approach can be then used to extend any other ON/OFF-based single-output minimizer for incomplete functions to a multi-output minimizer.

References

[1] R. Anderson "Programmable High-Speed State Machine with Sequencing Capabilities," *U.S. Patent*, no. 4,965,472, October, 1990.

[2] A. Coppola, M.A. Perkowski, R. Anderson, J. Freedman, and E. Pierzchala, "Synthesis of Very Fast Distributed Controllers Based on Tokenized State Machine Model," *Proc. IFIP Work. on Control-Dominated Synthesis,* Grenoble, pp. 1-15, September 1992.

[3] A. Coppola, R.E. Anderson, J. Freedman, and M.A. Perkowski, "VHDL Synthesis of Concurrent State Machines to a Programmable Logic Device", *VHDL Intern. User's Forum,* Scottsdale, Arizona, May 1992.

[4] R.K. Brayton, G.D. Hachtel, C.T. McMullen, and A.L. Sangiovanni-Vincentelli, "Logic Minimization Algorithms for VLSI Synthesis," *Kluwer Academic Publishers,* 1984.

[5] R.K. Brayton, P.C. Mc Geer, J.V. Sanghavi, and A.L. Sangiovanni-Vincentelli, "A New Exact Minimizer for Two-Level Logic Synthesis," in T. Sasao (ed), "Logic Synthesis and Optimization", *Kluwer Academic Publishers,* pp.1-31, 1993.

[6] M. Ciesielski, S. Yang, and M.A. Perkowski, "Multiple-Valued Minimization Based on Graph Coloring", *Proc. ICCD'89,* pp. 262-265, October 1989.

[7] Cypress Semiconductor, in *BIMOS CMOS Data Book*, 1992.

[8] Cypress Semiconductor, *WARP2 Documentation,* 1996.

[9] B.J. Falkowski, and M.A. Perkowski, "Algorithm for the Generation of Disjoint Cubes for Completely and Incompletely Specified Boolean Functions," *Intern. J. of Electr.,* Vol. 70, No. 3, pp. 533 - 538, March 1991.

[10] C. Files, Andre Reis, M. Perkowski, M. Robert, and D. Auvergne "Minimization of Networks with Complex Gates," *Workshop on Boolean Problems,* Freiberg, Germany, September 1996.

[11] D.S. Johnson "Worst-Case Behavior of Graph Coloring Algorithms," *Proc. of the 5th SE Conf. on Combinatorics, Graph Theory*

and Computing, Winnipeg, Canada, Utilitas Mathematica Publishing, pp. 513-528, 1974.

[12] C.J.H. McDiarmid, "Determining the Chromatic Number of a Graph," *SIAM J. Computing,* Vol. 8, pp. 1-19. 1979.

[13] R.E. Miller, *"Switching Theory",* Vol. 1 and 2, John Wiley, New York, 1965.

[14] L.B. Nguyen, M.A. Perkowski, and N.B. Goldstein, "PALMINI - Fast Boolean Minimizer for Personal Computers," *Proc. 24th DAC,* Miami, FL, pp. 615-621, June 1987.

[15] M.A. Perkowski, P. Wu, and K.A. Pirkl, "KUAI-exact: A new approach for multi-valued logic minimization in VLSI synthesis", *Proc. ISCAS*, pp. 401-404, May 1989.

[16] M.A. Perkowski, M.E. Chrzanowska-Jeske, E. Pierzchala, and A. Coppola, "An Exact Solution to the Fitting Problem in the Application Specific State Machine Device," *J. of Circuits, Systems, and Computers,* Vol. 4, No. 2, pp. 173-190, June 1994.

[17] M.A. Perkowski, and A. Coppola, "A State Machine PLD and Associated Minimization Algorithms", *Proc. of FPGA'92,* Berkeley, pp. 109-114, February 1992.

[18] M.A. Perkowski, T. Ross, D. Gadd, J.A. Goldman, and N. Song, "Application of ESOP Minimization in Machine Learning and Knowledge Discovery," *Proc. Reed-Muller'95*, pp. 102-109, September 1995.

[19] M. Perkowski, M. Marek-Sadowska, L. Jozwiak, T. Luba, S. Grygiel, M. Nowicka, R. Malvi, Z. Wang, and Jin S. Zhang, "Decomposition

of Multiple-Valued Relations," *Proc. ISMVL'97 conference*, pp. 13-18, May 1997.

[20] R. Rudell, and A. Sangiovanni-Vincentelli, "Exact Minimization of Multiple-Valued Functions for PLA Optimization," *Proc. ICCAD'86*, pp. 352-355, November 1986.

[21] T. Sasao, and M. Fujita (Eds.), "Representation of Discrete Functions," Kluwer Academic Publishers, 1996, chapter 13.

[22] K. Skahill, "VHDL for Programmable Logic," *Addison-Wesley Publ.*, 1996.

MCNC Examples	Variables		start from minterms		start from Espresso		start from Spectra disjoint		SOCMIN with	Time
	input	output	mntrms	CDEC	cubes	CDEC	cubes	CDEC	CSA	(sec)
5xp1	7	10	-	-	65	37	73	36	36	6.9
b12	15	9	-	-	43	27	52	27	27	8.7
con1	7	2	161	7	9	8	10	8	7	0.1
cu	14	11	-	-	19	19	19	19	19	2.1
f51m	8	8	-	-	76	20	76	20	20	5.8
i1	25	16	-	-	28	20	30	20	20	9.5
inc	7	9	305	29	30	29	34	34	29	8.5
misex1	8	7	547	12	12	12	14	12	12	2.0
misex2	25	18	-	-	28	27	28	27	27	1.7
rd53	5	3	42	26	31	29	31	27	26	1.4
rd73	7	3	185	20	127	22	127	20	20	1.4
rd84	8	4	665	30	255	30	255	30	30	2.7
sao	10	4	754	13	58	13	93	13	13	0.8
squar5	5	8	85	22	25	25	26	23	22	1.8
temp	5	3	33	7	13	7	16	7	7	0.3
vg2	25	8	-	-	110	38	219	36	36	11.4

Table 1

Multi output MCNC benchmarks. All CDEC covers calculated with SOCMIN. Graphs for coloring created from minterms (mnt), cubes from Espresso-minimized SOP, or cube covers calculated by respective minimizers. Column SOCMIN has results calculated using cubes produced by Algorithm 2. Time given for split cubes, it was counted together with the preprocessing time.

MCNC Examples	input literals	start from minterms # terms	start from minterms CDEC	start from Espresso # terms	start from Espresso CDEC	start from Spectra disjoint # terms	start from Spectra disjoint CDEC	SOCMIN with Algorithm 2	Time (sec)
5x01	7	76	4	7	4	7	4	4	0.03
5x5	7	88	6	10	7	10	7	6	0.09
bw10	5	28	2	3	3	3	3	2	0.03
bw18	5	11	4	6	5	6	5	4	0.04
bw20	5	18	4	6	5	6	5	4	0.05
duke10	22	408576	9	14	10	16	9	9	0.51
f501	8	244	13	23	14	23	14	13	0.45
f52	8	216	12	18	12	18	12	12	0.26
f53	8	192	8	14	9	14	9	8	0.16
f54	8	176	6	10	7	10	7	6	0.09
misex20	10	8	7	7	7	7	7	7	0.11
misex21	6	52	5	11	6	11	6	5	0.10
misex22	6	20	4	6	5	7	5	4	0.05
misex23	6	22	4	6	5	8	5	4	0.06
misex48	6	52	5	11	7	12	7	5	0.09
misex54	6	52	5	11	6	12	6	5	0.11
misex55	6	20	4	6	5	7	5	4	0.05
misex56	6	22	4	6	5	8	5	4	0.06
rd53	5	20	7	10	10	8	10	7	0.09
rd731	7	84	22	42	25	10	25	22	2.4
rd733	7	280	13	35	25	42	13	13	1.2
sao23	10	1664	9	22	9	31	10	9	0.76
sao24	10	673	9	21	12	33	9	9	0.85
z42	7	128	15	28	16	28	16	15	0.78

Table 2

Comparison of numbers of terms and numbers of CDEC implicants in covers of single output MCNC benchmarks.

Example	ON	Terms		Literals		DFC		Time
		Es	So	Es	So	Es	So	sec
add0	120	15	8	64	42	252	164	0.23
add2	128	16	8	68	44	268	180	0.22
add4	128	2	2	4	4	12	12	0.50
ch15f0	88	12	10	60	54	236	244	0.14
ch176f0	64	2	2	6	6	20	20	1.04
ch177f0	128	2	2	4	4	12	12	0.50
ch22f0	48	6	4	30	24	116	92	1.44
ch30f0	64	7	5	32	40	124	156	5.98
ch47f0	52	9	7	48	40	188	176	1.66
ch52f4	50	18	15	108	96	428	428	1.54
ch70f3	24	5	4	28	24	108	104	1.73
ch74f1	39	10	8	58	55	228	224	1.88
ch83f2	38	17	11	115	82	456	344	1.65
ch8f0	224	7	6	16	15	60	60	0.06
c_4_on	70	70	70	560	560	2236	2236	14.7
grt_th	120	15	8	64	42	252	168	0.38
intrvl1	58	16	13	96	90	380	380	3.49
intrvl2	128	22	14	110	92	436	380	4.09
kdd1	160	2	2	3	3	8	16	0.03
kdd10	120	8	4	28	18	108	68	0.09
kdd2	24	2	1	8	5	28	16	0.01
kdd3	80	2	2	5	5	16	24	0.02
kdd4	128	1	1	1	1	0	4	0.06
kdd5	106	4	4	13	13	48	13	0.11
kdd6	240	4	1	4	4	12	12	0.76
kdd7	175	4	4	8	8	28	28	0.85
kdd8	64	2	2	6	6	20	28	0.14
kdd9	64	4	4	36	24	140	100	0.76
maj_gate	93	56	20	280	151	1116	612	10.3
mdls_2	43	10	6	45	37	160	152	1.5
mux8	128	4	4	12	12	44	60	0.57
pal	16	16	16	128	128	508	508	0.30
pal_db_op	160	29	23	151	136	600	600	2.95
pal_ou	118	46	40	291	262	1156	1168	12.7
parity	128	128	128	1024	1024	4092	4092	22.8
remder2	88	23	16	137	118	528	492	2.97
rnd_m1	1	1	1	8	8	28	28	1.79
rnd_m10	10	9	9	71	71	280	280	2.15
rnd_m25	25	20	19	154	148	612	612	5.8
rnd_m5	5	5	5	40	40	156	156	1.9
rnd_m50	50	34	28	250	214	996	940	6.58
rnd1	122	50	39	324	302	1320	1232	11.5
rnd2	124	47	39	292	269	1164	1184	9.42
rnd3	134	49	32	306	240	1240	972	7.39
substr1	142	6	6	18	18	68	68	1.24
substr2	79	5	5	20	20	76	76	0.95
sbtrct1	104	34	22	200	145	796	616	4.87
sbtrct2	128	2	2	4	4	12	12	0.48

Table 3

Comparison of numbers of terms, numbers of literals, and DFC values for SOP expressions of Espresso (Es), and SOC expressions of SOCMIN (So), for single-output Machine Learning benchmarks. Time is given for SOCMIN.

example	25% don't cares				50% don't cares				90% don't cares			
	Espresso		Socmin		Espresso		Socmin		Espresso		Socmin	
	#t	DFC	#t	DFC	#t	DFC	#t	DFC	#t	DFC	#t	DFC
add0	27	620	6	196	26	668	6	124	13	408	3	44
add2	26	576	9	268	21	528	6	172	10	308	2	48
add4	17	332	2	12	17	400	2	12	10	304	2	40
ch15f0	24	592	10	340	22	604	10	344	8	252	3	112
ch176f0	11	232	2	20	10	248	2	20	4	116	1	0
ch177f0	22	456	2	12	10	208	2	12	6	184	1	0
ch22f0	12	292	4	156	12	332	4	164	5	156	2	28
ch30f0	15	352	5	136	15	388	5	164	5	156	2	52
ch47f0	16	408	7	192	14	380	8	216	4	124	3	104
ch52f4	17	440	15	408	14	392	12	360	3	92	1	40
ch70f3	8	208	3	84	6	164	2	64	1	28	1	28
ch74f1	12	296	6	248	10	264	5	188	4	124	1	24
ch83f2	15	420	13	384	12	344	7	284	2	60	1	16
ch8f0	38	816	32	672	36	860	6	184	18	548	3	68
c_4_ones	50	1596	27	1320	33	1052	14	628	4	124	2	36
grt_than	28	628	6	188	31	776	10	264	12	372	2	28
intrvl1	15	368	12	324	13	332	8	236	6	180	1	0
intrvl2	27	612	13	468	27	692	9	248	10	300	3	48
kdd1	24	492	2	12	23	544	2	16	11	324	2	20
kdd10	28	640	6	160	25	652	4	112	7	204	2	44
kdd2	4	84	3	72	4	96	1	28	1	24	1	24
kdd3	14	312	2	28	14	336	2	28	6	176	1	0
kdd4	20	408	14	376	22	532	1	0	10	304	1	0
kdd5	19	400	4	76	23	588	4	48	7	216	3	88
kdd6	33	656	23	620	37	840	1	0	17	520	1	20
kdd7	31	664	22	644	29	676	5	64	13	400	3	60
kdd8	16	388	12	372	18	484	2	20	8	252	1	0
kdd9	18	448	16	320	15	404	4	144	6	184	2	24
maj_gate	44	1056	12	456	29	776	10	332	10	308	2	52
mod2	12	296	5	156	12	320	4	128	5	156	1	4
mux8	24	516	4	56	21	512	4	68	7	212	1	0
pal	14	444	14	444	12	380	6	268	2	60	1	12
pal_dbl_op	41	992	40	952	35	936	11	352	12	364	3	36
pal_op	42	1108	39	992	34	960	15	568	10	304	2	48
parity	94	3004	20	596	63	2012	14	544	8	252	2	44
remder2	28	720	13	500	23	608	10	404	5	152	2	36
rnd1	46	1228	44	1056	36	1004	18	824	12	372	4	60
rnd2	35	988	15	696	32	888	25	776	11	340	3	52
rnd3	48	1276	24	1076	37	1024	15	552	11	340	3	52
rnd_m1	0	0	1	0	1	0	1	0	1	0	1	0
rnd_m10	6	188	6	188	2	60	2	60	1	0	1	0
rnd_m25	15	468	12	432	8	248	6	196	1	28	1	28
rnd_m5	2	60	2	60	1	28	1	28	1	0	1	0
rnd_m50	27	804	24	744	18	540	9	436	3	92	1	40
sbstr1	28	620	7	104	30	784	9	200	11	340	4	116
sbstr2	20	472	5	108	15	384	5	136	7	216	2	48
sbtrct1	43	1160	15	536	31	868	9	340	10	312	0	0
sbtrct3	18	356	2	12	16	368	2	12	9	268	2	16

Table 4

Comparison of numbers of terms and DFC values for SOP expressions of Espresso and SOC expressions of SOCMIN, for single-output Machine Learning benchmarks in which 25%, 50%, and 95% of don't cares were next randomly generated (cares randomly removed)