# NEW APPROACH TO LEARNING NOISY BOOLEAN FUNCTIONS

MAREK PERKOWSKI, LECH JOZWIAK +, and SANOF MOHAMED *
*Department of Electrical Engineering, Portland State University, Portland, OR 97207-0751*
*Portland, Oregon 97207-0751, USA, Tel: 503-725-5411, 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, lech@eb.ele.tue.nl*
*COMPASS Design Automation, 1865 Lundy Ave, M/S 430, San Jose CA 95131, Tel: 408-434-7893, sanof@compass-da.com*

We give a new formulation of noise removal from data being Boolean functions used in Conceptual Inductive Learning. The algorithm is used as a part of a functional decomposition program. Paper gives an efficient heuristic algorithm for the minimization of multi-output Exclusive DNF (EDNF) expressions that generalize the Disjunctive Normal Forms (DNF) expressions popularly used in Machine Learning (ML). The algorithm can be used for preprocessing, after any stage of decomposition, and for postprocessing. In EDNF, the ANDs of DNF are AND-ed with NANDs, providing Exclusion cases. The EDNF is thus a disjunctive sum of AND/NAND gates, called Conditional Decoder (CDEC) functors. A pruning technique is based on analysis of sizes of AND and NAND parts in CDEC functors, and the numbers of true and false minterms covered by them. Experiments on ML benchmarks prove that our approach generates high-quality solutions, is especially efficient on strongly unspecified functions, and reduces the recognition errors.

## 1 Introduction.

In recent years there has been a big progress in Inductive Learning approach to Machine Learning [5]. The traditional machine learning approaches such as Neural Nets (NN) postulated that the recognizing network had some assumed structure which was "tuned" by the learning process (for instance, by decreasing and increasing the numerical values of some coefficients). Thus, creating a new structure was accomplished by setting some coefficients to zero. All "new" structures are in a sense "hidden" in the assumed mother-structure. Also the type of an element, such as a formal neuron in Neural Nets, is decided before the learning takes place. Thus, the approach has inherent bias, and explanation of learned concepts is also usually not possible. In contrast, the approach of conceptual inductive learning is the *concept learning from examples* (called also *concept acquisition*). The task is to induce general descriptions of concepts from specific instances of these concepts. Various logic structures have been used for descriptions: DNFs (AQ, CN, Espresso-Dexact, Espresso), Vari-

able Valued Logic expressions (Michalski), decision trees (Quinlan, C4.5), and other [5,13]. Such systems can produce better explanations than NNs and reduce recognition errors. However, these systems still assume certain structure of the expressions derived in the inductive process. *Constructive Induction* is the induction process that changes the description space, it produces new descriptors (called also *features* or *concepts* that were not present in the input events [5]. Few Constructive Induction systems have been recently developed [5], and they provide even better explanations. In this paper we are focusing on the Constructive Induction system based on Functional Decomposition [4,14,11,10,16]. Although the very early ideas of such systems based on decomposition come perhaps from the famous checkers program by Samuel, the first group who formulated the learning problem as Ashenhurt Decomposition were Biermann et al [1]. Unfortunately, their paper did not make influence on Machine Learning community and is very rarely referenced. In [14] a machine learning system by Ross, based on logic synthesis methods, is described. The main component of this system is an algorithm performing Ashenhurst/Curtis decomposition (ACD) of Boolean functions. There is no a priori assumption of structure of the learned description, nor on the type of the elements. The elements, called *blocks*, are arbitrary discrete mappings (functions). Blocks are connected to form a network with no loops. Both the structure and the elements are **calculated** in the learning process, and this process is entirely based on finding patterns in data. In [11,10] a Machine Learning system that generalized the approach of Ross by formulating the theory of non-disjoint decomposition of Multi-Valued, Multi-output Relations has been presented. Observe, that ML data benchmarks are naturally such relations. Pattern finding is in our system a generalization and formalization of feature extraction. Furthermore, it constructs this network representation by minimizing the complexity, as in Occam-based learning - we use the DFC measure of complexity defined in [14]. Similar systems are also described in [4,16].

It has been shown [14,16,4,11,10] that the decomposition approach leads to better results (a smaller classification error, a solution being the description simpler and easier to understand) than the competing approaches of Neural Nets, tree classifiers and logic methods. Unfortunately, for large functions the decomposition method is slow, because the NP-complete problem of creating and coloring a graph of columns incompatibility must be solved a lot of times, and on large data. Also, this approach, called noise-free decomposition, assumed perfect (non-noisy) data. In order to extend this approach to realistic data that have continuous values of attributes, missing and false attributes, and false decisions, as well as data coming from unreliable sources, we developed an improved decomposition, called "noisy decomposition", that takes all

these "noisy" factors into account.

Although our noisy decomposition theory is for multivalued, multioutput relations and includes three stages: preprocessing, (iterated) decomposition step and postprocessing, for simplification, this paper discusses only binary, single-output functions (The full theory is presented in [12]. The case of multi-output functions is discussed in [7]). Similarly, we will not discuss in detail the noisy column-compatibility approach applied during the decomposition step, but will concentrate only on the three-level network (EDNF) minimization used in preprocessing or after every decomposition stage.

The proposed method of dealing with noisy functions can be used not only as a part of the decomposition, but by itself, as a stand-alone Machine Learning method, similar to DNF approaches (but this would no longer be a method of Constructive Induction). While this paper presents a stand-alone approach to deal with noisy functions, our next paper [12], will discuss the total system with both approaches combined, and will demonstrate the effect of noise removal on the recognition errors of the entire noisy decomposition.

## 2 The problem of noisy functions

Input variables are called attributes, output variables are called decisions. A minterm (called also a sample or data instance) of single-output function is the vector of attribute values and the decision value. If the decision value is **true** (1, or "belongs to the class") then the minterm is called a *true minterm). If the decision value is* **false** *(0, or "does not belong to the class") then the minterm is called a* false minterm). Thus, for a function with three variabls, a true minterm $< a_1, a_2, a_3, d_1 > = < 0, 1, 1; 1 > = 0111$ is represented as a value 1 in Kmap in cell $< a_1, a_2, a_3 > = 011$. A false minterm 1000 is represented as a value 0 in cell 100. (Output) don't care is a vector of attribute values for which the function value is not specified. True and false minterms are called *cares.*

ML benchmark "functions" (data for learning) can have the following properties which distinguish them from the data (Boolean functions) used in circuit design:

(1) **Strongly Unspecified**. ML problems have an extremely high (more than 99%) percent of don't cares. The missing, or unknown decision data can be represented as don't cares. This is in contrast to circuit design where the percent of don't cares is rarely higher than 50%.

(2) **Representation**. The data are usually minterms or (rarely) cubes. This is in contrast to circuit design where the initial data are netlists, BDDs or cubes.

(3) **Multi-Output.** Sometimes classification of patterns into more than two categories is desired (one wants not only to distinguish a "friend from foe" airplane, but also to learn its orientation, speed, etc.). In terms of logic synthesis, this property corresponds to concurrent minimization of switching functions with many outputs. (f - binary variable {friend, foe}, g - 10-valued variable "orientation", h - 20-valued variable "speed").

(4) **Many input variables.** Practical ML-related minimization problems require at least 30 binary input variables but more typically, about 100 multiple-valued variables. In machine learning benchmarks, with the increase in the number of input variables, there is only a small increase in the number of both true and false minterms, but a dramatic increase in the number of don't cares. For instance, it is reasonable to expect that for a function of 100 attributes there will be no more than 10,000 cares.

(5) **Binary, symbolic, numerical, multi-valued or continuous attributes**. Attributes in ML problems are naturally Boolean, multiple-valued, symbolic, or continuous. Symbolic values can be converted to multiple-valued. Continuous values can be be discretized, and thus converted to multiple-valued. Any value of a multiple-valued attribute can be encoded by a vector of binary variables. Although this is not the best approach [10,11], for simplification, we will consider here the encoding, and thus binary functions.

(6) **Noise in attributes.** In the context of ML, "noise in the attribute" means that the measurement is imprecise or subjective evaluation by the person collecting information is wrong (*"is this actress dark-blond or red-hair?"*) This can be modelled by taking some known function and next adding some statistical noise to the values of attributes. "Noise" flips a correct bit to an incorrect value. Thus, minterm 0110 becomes for instance 1110 because of noise in the first attribute.

(7) **Noise in decisions.** This noise is even more dangereous than the noise in attributes [13,5]. For instance, a typist who typed information to the data-base can make a mistake and encode a minterm corresponding to a patient without cancer (correct decision 0) as one who has cancer (incorrect decision 1).

(8) **Unknown values of attributes.** The data in the form can be left missing, for instance the age group is omitted for some reason from the data about a lung cancer patient. In a binary case, a single unknown in a minterm means that a symbol corresponding to this minterm is placed in one of two positions on the Kmap. For instance, if the second attribute is unknown, represented by minterm 0?11, value 1 should be placed either in cell 001 or in cell 011. This possibility of choice should be taken into account by the learning program.

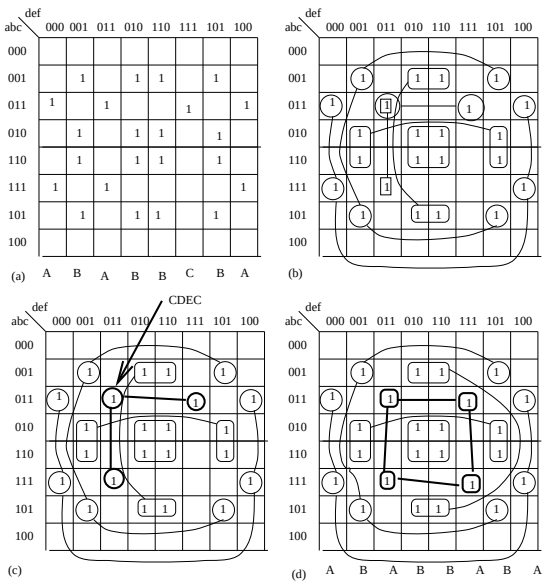


Figure 1: *(a) Function $f$ to Example 1. Types of columns, A,B,C, are below the table; (b) DNF for function $f$, (c) original EDNF for function $f$, (d) pruned EDNF for function $f$ with CDEC-implicant $bcef$ replaced with prime implicant $bcef\overline{ad}$. False Minterm 111111 is replaced with true minterm. Types A,B of the new columns are below the table.*

(9) **Data with conflicts, from unreliable sources, data with repeated minterms.** If one has two minterms with the same values of attributes and different values of decision, there is clearly a conflict. Such data can be simply discarded (treated as a don't care), but it is better to count how many times the decision value for a given cell was 1 and how many times 0, and utilize these information in the decision making by assigning a valued of *confidence factor* to the minterm corresponding to this cell. Similarly, if there exist repeated non-conflicted data, their numbers of occurences may be counted. Thus, a decision 1 that repeated many times in some cell is more reliable than a 1 that occured only once in this cell. Similarly, if the data come from various people collecting and preparing data bases, or from various measuring devices, some of them may be more reliable than others, and thus confidence factors are added to each minterm. Concluding, each minterm can have an additional numerical value in interval [0.0 1.0] that denotes the probability of having the respective value of the decision. Similar confidence factor values can be associated with every value of an attribute.

(10) **Discretization problems.** The effect of discretization of continuous attributes is somewhat similar to the unknown. For instance, for a continuous attribute, the value 3.5 can be discretized to either value 3 or value 4, but not to both. Thus, after discretization of the second attribute to three bits, the two-argument true minterm $< 0, 3.5, 1 >$ should be the value 1 put to cell $< 0, 011 >$ or value 1 put to cell $< 0, 100 >$. This is similar to the unknown value.

(11) **Relations.** Relations may be useful to model many problems with imprecise data [10,11]. For instance, discretization of a decision value can be described as a relation (continuous value 3.5 can be discretized to multiple-valued value 3 or 4, thus a relation of input vector with both decision values 3 and 4 is created). Here, we will assume that we deal with functions, i.e. special cases of relations. Dealing with noise in relations is presented in [12].

## 3 An approach to Noise Removal in Noisy Decomposition.

*Example 1.* Given is a function from Fig. 1a. One can see, that there are three types of columns, A,B,C, so the column compatibility index $\mu$ for bound set of input attributes $d, e, f$ is three. This value of $\mu$ would be used in a standard non-noisy decomposition program, and thus two new intermediate variables would be necessary for a Curtis decomposition with $\mu = 3$. Let us, however, observe, that if we would change just one cell abcdef = 111111 from value 0 to value 1, then there would be only two types of columns, column $def$ would become of type A, $\mu$ would decrease to 2, and thus Ashenhurst decomposition

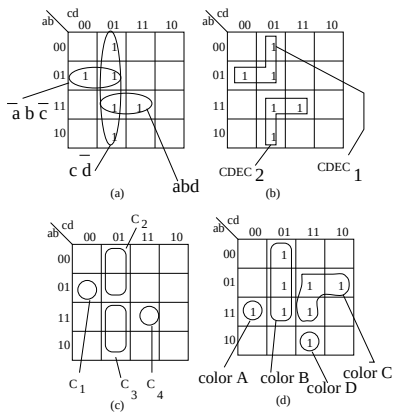


Figure 2: *(a,b) Function that demonstrates that the minimum EDNF cannot be obtained by factorization of primes, (c) Compatible and conditionally compatible cubes from Example 2. Cubes $C_2$ and $C_3$ are compatible, cubes $C_1$ and $C_2$ are conditionally compatible, (d) function to Example 5 and its CDEC cover.*

with only one intermediate variable and smaller complexity becomes possible: $f = bc \oplus (b + c)x$; $x = e \oplus f$. Let us observe, that a new concept $x$ has been induced by this decomposition stage. Also, variables $a$ and $d$ become *vacuous*, which means that these attributes were irrelevant to the concept induced. Decomposition stage reduced the DFC complexity of the function. Observe also that the reduction in complexity is more substantial in this noisy decomposition than it would be in the noise-free decomposition with $\mu = 3$. The method to deal with noise is thus to perform such minterm value replacements that the column multipicity will be decreased by the minimum number of changes. We assume that that only very few cells can be modified like this - this technique is similar in principle to the prunning techniques used for decision trees [13,5]. If the number of necessary changes would exceed some threshold, the modification would be not performed. This method can be applied at every stage of decomposition.

*Example 2.* Another method is to create a DNF (for the initial function, or the function from any block of the decomposition), next remove from DNF those products, that cover very few true minterms, and allow to have products that cover few false minterms. Good selection of products that would allow for such function modifications is, however, a difficult task. We propose an approach based on the new concept of Exclusive DNF (EDNF) ex-

pressions that generalize the Disjunctive Normal Forms (DNF) expressions. The function from Fig. 1a can be minimized to a DNF as in Fig. 1b: $bcef + b\bar{c}\bar{e}f + \bar{b}c\bar{e}f + bc\bar{e}\bar{f} + \bar{b}ce\bar{f}$. If we, however, allow the false minterm 111111 to become a true minterm, then the DNF from Fig. 1d would be found. The effect on the noise-free decomposition in this case would be the same as in the noisy column compatibity method from Example 1. Such method can be used as a preprocessing, postprocessing and/or after each stage of a decomposition. The decomposition itself can be noise-free or can use the method from Example 1. The numbers of false minterms treated as true minterms can be user-controlled, similarly as in prunning methods in C4.5 or other decision diagram creating programs. Analysis of possibe noise sources as described in section II (points (6) - (10) and use of confidence factors are also performed to make all prunning decisions. We assume that the groups of 0's should be treated as 1's especially if they are located closely and not separated by 1's. In such situation, sub-functions of type $P \cdot \overline{Q}$ are created, where $P$ and $Q$ are product terms. Part $Q$ is the *exclusion product*. $P \cdot \overline{Q}$ is called a *CDEC-implicant*[7]. We assume that product $P$ does not include 0's. If the exclusion product $P$ includes few 0's, it can be replaced with a constant 1, thus creating a product implicant $Q$. In our case, the DNF from Fig. 1d was obtained by finding EDNF from Fig. 1c and next replacing CDEC-implicant $bcef\overline{ad}$ with prime implicant $bcef$. We assumed that if $Q$ includes one 0 than it can be dropped (in practize only large cubes $P$ with few 0's and many don't cares are replaced with 1).

Similarly, CDEC-implicants that cover few 1's are discarded. Again, confidence factors are used. as well as sizes of $P$ and $Q$ are being considered by the heuristic evaluation rules Our method minimizes the function to EDNF form only once, and next the minimized EDNF expression is transformed several times by heuristic rules with respect to all noise-related information (6) - (10). Such approach leads to several EDNF expression variants of reduced DFC complexity, that are next verified towards function benchmarks to find those with minimized error. Our EDNF method to deal with noise allows to treat separately false positive (0 replaced with 1) and false negative decisions (1 replaced with 0), which is useful in some decision problems, especially in medical applications. Please note, that this entire approach is based on Occam Principle that a simple function (small DFC) is more probable than a complex one.

Because we want to create as large as possible CDEC-implicants, the concept of *prime CDEC-implicants*, i.e. CDEC-implicants that are not totally included in any other CDEC-implicants, is useful. From these definitions, a product and prime implicants used in DNF minimization are special cases of

CDEC-implicants, but a prime is not necessarily a prime CDEC, because it can be included in a CDEC-implicant. EDNF minimization is based on covering with *CDEC-implicants*, which we will call *CDEC-covering*. A CDEC-implicant is an implicant that can be realized with a single CDEC gate. It has an *AND-part* and a *NAND-part* sometimes called an *OR-part*, since $ab\overline{cd} = ab(\bar{c} + \bar{d})$. The concept of a CDEC-implicant is a concept generalization of a prime implicant (a prime) and a product implicant[8,3] used in standard DNF minimizing programs such as Espresso[15].

## 4 Conditionally Compatible Cubes and Compatible Graph Coloring for EDNF Minimization

Boolean functions are implemented as sets of minterms or cubes. (Minterms can be grouped to cubes in preprocessing to decrease the problem size. Many benchmarks have cubes and changing them to minterms would create files too large to handle efficiently.) 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. A **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. Standard notions of literals, sum of products, essential and secondary essential prime implicants[9], consensus, sharp, disjoint sharp, intersection and Hamming distance of cubes will be used by our program. 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*. 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 $C_j$. For instance; for $C_i = X101$ and $C_j = X110$, $C_i \uplus C_j = X1XX$ is the supercube of $C_i$ and $C_j$. We say that *two cubes overlap* if they have a non-empty intersection cube. 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): $(C_i \uplus C_j) \sqcap OFF(f) = \emptyset \Rightarrow$ *$C_i$ and $C_j$ 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 0). 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 $SOFFP(SU_{ij})$. 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 cube $SOFFP(SU_{ij})$ as the NAND-part. The cube $SOFFP(SU_{ij})$ is called the **condition cube** under which $C_i$ and $C_j$ are conditionally compatible. Two cubes are **incompatible** if they are not compatible nor conditionally compatible. Such cubes cannot be combined to a single CDEC-implicant. The set $\Pi$ 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 $\Pi$ can be described by a single CDEC-implicant: $(AND\ part, NAND\ part) = (\biguplus_{C_i \in \Pi} C_i\ ,\ SOFFP(\biguplus_{C_i \in \Pi} C_i))$

*Example 3.* The Boolean function $f$ (Fig. 2a) 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. 2c). 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 $\{C_3, C_4\}$ (see Fig. 3a). 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.

EDNF minimization is based on *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 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 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

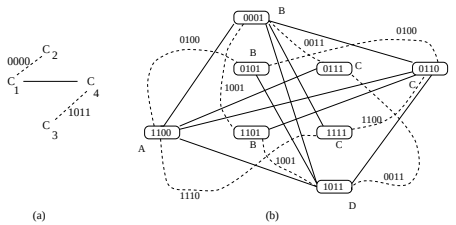


Figure 3: *Compatible Coloring: (a) graph GCCC for Examples 3 and 4, (b) graph GCCC for Example 5.*

*conditional edge* if $MI_i$ and $MI_j$ are compatible under the condition indicated by the label 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 groups of nodes have the same label on all its conditional edges, the nodes are compatible without any condition, 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.

*Example 4.* Using the minimal conditional coloring method to the function from Fig. 2a a GCCC with nodes $C_1$, $C_2$, $C_3$, and $C_4$ is created (Fig. 3a). The conditionally compatible nodes are shown by dotted edges connecting them. Conditions are written near edges. The coloration 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 with only two CDEC gates (Fig. 2b). As we see, the minimal solution can be obtained by the factorization of product implicants that are not primes: $f = \bar{a}b\bar{c} + \bar{a}\bar{c}d + a\bar{c}d + abd = \bar{a}\bar{c}(b+d) + ad(\bar{c}+b) = \bar{a}\bar{c}\overline{(\bar{b}\bar{d})} + ad\overline{(c\bar{b})} = CDEC_1 + CDEC_2$. This function demonstrates, that the approach based on factoring prime implicants can not find the minimum CDEC cover. 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 three CDEC gates and the CDEC-implicants would be identical to the prime implicants. This example shows that prime CDEC-implicants must be created not only from prime implicants, but also from some product implicants *included* in prime implicants.

*Example 5.* Fig. 2d illustrates a minimum color cover with CDEC-implicants in a Karnaugh map of certain function $g$. These CDEC-implicants were created by CDEC-compatible coloring of the nodes of the graph from Fig. 3b. In Figs. 2d and 3b color A describes a CDEC-implicant $ab\overline{\bar{c}d}$. Color B corresponds to

a CDEC-implicant $\overline{c}da\overline{\overline{b}}$. Color C corresponds to a CDEC-implicant $bca\overline{\overline{d}}$. Color D corresponds to a CDEC-implicant $a\overline{b}cd$. The nodes of the graph in Fig. 3b correspond to the minterms in the Karnaugh-Map from Fig. 2d. 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. The solution found, $g = ab\overline{c}\overline{d} + \overline{c}da\overline{\overline{b}} + bca\overline{\overline{d}} + a\overline{b}cd$, 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.

In Example 5 all groups of nodes with the same color have only one type of label on edges connecting them, but in general they can have many types of labels. 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 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, $l(SC_i) \uplus \ldots \uplus l(SC_{i+k})$, does not intersect any nodes from the set $\{MI_j, \ldots, MI_{j+l}\}$. Compatible coloring is one in which any set of nodes colored with the same color is a compatible set of nodes (compatible set of cubes). Compatible set of nodes is a set in which all nodes correspond to a set of compatible nodes. The problem of minimal conditional graph coloring is to find such coloring that is compatible, minimizes the number of colors, and minimizes the literal cost as the secondary cost function. The literal minimum solution is the CDEC-minimum solution which in addition has the minimum literal cost.

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*. For instance, multi-coloring of nodes for a function from Example 5 would create, among others, a solution $g" = \overline{c}da\overline{\overline{b}} + abc\overline{\overline{d}} + cd\overline{a}\,\overline{\overline{b}} + a\overline{b}\overline{\overline{c}}\,\overline{\overline{d}}$. The multi-coloring method creates larger CDEC-implicants. Observe that the minimum literal cost solution $g'$ from Example 5 could not be found from a cover with prime CDEC-implicants, since all the primes from this solution are included in some prime CDEC-implicants. The solution with the exact minimum number of

| MCNC Examples | Variables | | from minterms | | from Espresso | | from split | T |
|---|---|---|---|---|---|---|---|---|
| | in | ou | mnt | CD | cu | CD | CD | sec |
| 5xp1 | 7 | 10 | - | - | 65 | 37 | 36 | 6.9 |
| b12 | 15 | 9 | - | - | 43 | 27 | 27 | 8.7 |
| con1 | 7 | 2 | 161 | 7 | 9 | 8 | 7 | 0.1 |
| cu | 14 | 11 | - | - | 19 | 19 | 19 | 2.1 |
| f51m | 8 | 8 | - | - | 76 | 20 | 20 | 5.8 |
| i1 | 25 | 16 | - | - | 28 | 20 | 20 | 9.5 |
| inc | 7 | 9 | 305 | 29 | 30 | 29 | 29 | 8.5 |
| misex1 | 8 | 7 | 547 | 12 | 12 | 12 | 12 | 2.0 |
| misex2 | 25 | 18 | - | - | 28 | 27 | 27 | 1.7 |
| rd53 | 5 | 3 | 42 | 26 | 31 | 29 | 26 | 1.4 |
| rd73 | 7 | 3 | 185 | 20 | 127 | 22 | 20 | 1.4 |
| rd84 | 8 | 4 | 665 | 30 | 255 | 30 | 30 | 2.7 |
| sao | 10 | 4 | 754 | 13 | 58 | 13 | 13 | 0.8 |
| squar5 | 5 | 8 | 85 | 22 | 25 | 25 | 22 | 1.8 |
| temp | 5 | 3 | 33 | 7 | 13 | 7 | 7 | 0.3 |
| vg2 | 25 | 8 | - | - | 110 | 38 | 36 | 11.4 |

Table 1: *Multi output MCNC benchmarks. All CDEC covers (CD) calculated with Socmin. Graphs for coloring created from minterms (mnt) or cubes from Espresso-minimized DNF, Column Socmin has results calculated using split cubes. Time (T) given for split cubes, it was counted together with the preprocessing time.*

CDEC-implicants is called CDEC-minimum solution. The exact minimization of the number of CDEC-implicants is based on the theorem that states that if 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 coloring), then the CDEC covering created from this coloring has the exact minimum number of CDEC-implicants.

The CDEC cover minimization process results in a EDNF. The exact EDNF representation of function $f$ has never more terms than a DNF of this function because the set of all CDEC implicants includes the set of prime implicants. Observe that we do not generate all primes nor all prime CDEC-implicants, but are still able to find the exact cover. 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 [2,15]. A similar property can be shown for prime CDEC-implicants. 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 EDNF with the minimum literal cost. In addition, the covering problem is NP-hard. The covering table can become too large to store in memory. Therefore, our approach is based on graph coloring - one NP-hard problem is solved instead of two, and a minimum literal solution is always found. The graph GCCC for coloring can be created with arbitrary cubes from the ON-set of the function $f$ as nodes. For instance, these can be: minterms, primes, minimal product implicants

[8,3], or the (optimal) split cubes used here. They are based on splitting large primes to smaller cubes that can be next recombined to CDEC-implicants. 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 an improved execution speed, but the minimum EDNF can be lost. Our coloring algorithm can create the graph from any type of cubes. We use special kind of cubes called *split cubes*, for their efficiency. This means that the minimal solution can be in theory lost, but experimental results show that this never happened on the tested by us benchmarks.

## 5 Experimental Results

We created a program, called Socmin, and tested it extensively on MCNC and Machine Learning benchmarks. Let us observe that contrary to MCNC 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). A Miller method [6] for transforming a multi-output problem to a single-output problem has been used in order to extend the presented approach to multi-output functions. We verified all our results. Some results for benchmark functions are presented in Tables 1, 2 and 3. The goals of the presented here experiments with Socmin were to answer the following questions: How much improvement in implicants, literals or DFC is gained by using the CDEC-implicants instead of prime implicants? How fast is Socmin? How large functions can be minimized? Table 1 has columns for input and output variables, for experiments with various types of initial cubes Minterms, Espresso, and CS (*Cube Split Algorithm*), and for Time. Column *from Minterms* has two subcolumns; *mnt*, which includes the number of minterms, and *CD*, which includes the number of CDEC-implicants in exact solutions generated from minterms by Socmin. Column *from Espresso* has two subcolumns; *cu* and *CD*. 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 EDNF generated by Socmin from primes of Espresso as the starting point for coloring. Finally, column "from split" shows the numbers of implicants in CDEC cover found by Socmin in a GCCC created from split cubes produced in preprocessing. 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

| 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 2: *Comparison of numbers of terms, numbers of literals, and DFC values for DNF expressions of Espresso (Es), and EDNF expressions of Socmin (So), for single-output Machine Learning benchmarks. Time is given for Socmin.*

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 EDNF is substantially smaller than the DNF - 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. In multi-coloring mode the nodes can be colored with more than one color, which further decreases the number of literals. 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 2, 3). The numbers of terms, literals, and DFC were compared. 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 3 presents these results with 25%, 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; for both single-output and multi-output functions, EDNFs are better than DNFs, Socmin produces high quality EDNFs, and is very fast.

| example | 25% don't cares | | | | 90% don't cares | | | |
|---|---|---|---|---|---|---|---|---|
| | Espresso | | Socmin | | Espresso | | Socmin | |
| | t | DFC | t | DFC | t | DFC | t | DFC |
| add0 | 27 | 620 | 6 | 196 | 13 | 408 | 3 | 44 |
| add2 | 26 | 576 | 9 | 268 | 10 | 308 | 2 | 48 |
| add4 | 17 | 332 | 2 | 12 | 10 | 304 | 2 | 40 |
| ch15f0 | 24 | 592 | 10 | 340 | 8 | 252 | 3 | 112 |
| ch176f0 | 11 | 232 | 2 | 20 | 4 | 116 | 1 | 0 |
| ch177f0 | 22 | 456 | 2 | 12 | 6 | 184 | 1 | 0 |
| ch22f0 | 12 | 292 | 4 | 156 | 5 | 156 | 2 | 28 |
| ch30f0 | 15 | 352 | 5 | 136 | 5 | 156 | 2 | 52 |
| ch47f0 | 16 | 408 | 7 | 192 | 4 | 124 | 3 | 104 |
| ch52f4 | 17 | 440 | 15 | 408 | 3 | 92 | 1 | 40 |
| ch70f3 | 8 | 208 | 3 | 84 | 1 | 28 | 1 | 28 |
| ch74f1 | 12 | 296 | 6 | 248 | 4 | 124 | 1 | 24 |
| ch83f2 | 15 | 420 | 13 | 384 | 2 | 60 | 1 | 16 |
| ch8f0 | 38 | 816 | 32 | 672 | 18 | 548 | 3 | 68 |
| c_4_on | 50 | 1596 | 27 | 1320 | 4 | 124 | 2 | 36 |
| grt_th | 28 | 628 | 6 | 188 | 12 | 372 | 2 | 28 |
| intrvl1 | 15 | 368 | 12 | 324 | 6 | 180 | 1 | 0 |
| intrvl2 | 27 | 612 | 13 | 468 | 10 | 300 | 3 | 48 |
| kdd1 | 24 | 492 | 2 | 12 | 11 | 324 | 2 | 20 |
| kdd10 | 28 | 640 | 6 | 160 | 7 | 204 | 2 | 44 |
| kdd2 | 4 | 84 | 3 | 72 | 1 | 24 | 1 | 24 |
| kdd3 | 14 | 312 | 2 | 28 | 6 | 176 | 1 | 0 |
| kdd4 | 20 | 408 | 14 | 376 | 10 | 304 | 1 | 0 |
| kdd5 | 19 | 400 | 4 | 76 | 7 | 216 | 3 | 88 |
| kdd6 | 33 | 656 | 23 | 620 | 17 | 520 | 1 | 20 |
| kdd7 | 31 | 664 | 22 | 644 | 13 | 400 | 3 | 60 |
| kdd8 | 16 | 388 | 12 | 372 | 8 | 252 | 1 | 0 |
| kdd9 | 18 | 448 | 16 | 320 | 6 | 184 | 2 | 24 |
| maj_gate | 44 | 1056 | 12 | 456 | 10 | 308 | 2 | 52 |
| mod2 | 12 | 296 | 5 | 156 | 5 | 156 | 1 | 4 |
| mux8 | 24 | 516 | 4 | 56 | 7 | 212 | 1 | 0 |
| pal | 14 | 444 | 14 | 444 | 2 | 60 | 1 | 12 |
| pal_db_op | 41 | 992 | 40 | 952 | 12 | 364 | 3 | 36 |
| pal_op | 42 | 1108 | 39 | 992 | 10 | 304 | 2 | 48 |
| parity | 94 | 3004 | 20 | 596 | 8 | 252 | 2 | 44 |
| remder2 | 28 | 720 | 13 | 500 | 5 | 152 | 2 | 36 |
| rnd1 | 46 | 1228 | 44 | 1056 | 12 | 372 | 4 | 60 |
| sbstr1 | 28 | 620 | 7 | 104 | 11 | 340 | 4 | 116 |
| sbstr2 | 20 | 472 | 5 | 108 | 7 | 216 | 2 | 48 |
| sbtrct1 | 43 | 1160 | 15 | 536 | 10 | 312 | 0 | 0 |
| sbtrct3 | 18 | 356 | 2 | 12 | 9 | 268 | 2 | 16 |

Table 3: *Comparison of numbers of terms and DFC values for DNF expressions of Espresso and EDNF expressions of Socmin, for single-output Machine Learning benchmarks in which 25%, and 95% of don't cares were next randomly generated (cares randomly removed)*

## 6 Conclusions

We presented a new approach, called Exclusive Disjunctive Normal Forms (EDNF) minimization to deal with noise in Machine Learning. The respective minimization algorithm has been developed and compared to known DNF minimizers. The method gives very good solutions, better than Espresso, for multi-output strongly unspecified functions of Machine Learning and Knowledge Discovery from Databases. Practical usefulness of EDNFs to reduce the number of terms and recognition errors has been demonstrated. More details on the algorithms and experimental results can be found in [7,12,10,11].

## 7 References


1. A.W. Biermann, J.R.C. Fairfield, T.R. Beres, "Signature Table Systems and Learning," *IEEE Trans. on Syst. Man and Cybern.*, Vol. 12, No. 5, pp. 635-648, 1982.
2. 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.
3. M. Ciesielski, S. Yang, and M.A. Perkowski, "Multiple-Valued Minimization Based on Graph Coloring", *Proc. ICCD'89,* pp. 262-265, 1989.
4. T. Luba, "Decomposition of Multiple-Valued Functions," *Proc. 25th ISMVL*, pp. 256-261, 1995.
5. Michalski et al (ed), "Machine Learning. An Artificial Intelligence Approach," Volumes 1-4, *Morgan Kaufmann Publishers*, San Mateo, California.
6. R.E. Miller, *"Switching Theory",* Vol. 1 and 2, John Wiley, New York, 1965.
7. S. Mohamed, M. Perkowski, L. Jozwiak, "Fast approximate minimization of multi-output Boolean functions in Sum-of-Product-Condition-Decoders structures," *Proc. Euromicro'97*, Sept. 1997.
8. 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, 1987.
9. 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, 1989.
10. M. Perkowski, T. Luba, S. Grygiel, M. Kolsteren, R. Lisanke, N. Iliev, P. Burkey, M. Burns, R. Malvi, C. Stanley, Z. Wang, H. Wu, F. Yang, S.

Zhou, and J. S. Zhang, "Unified Approach to Functional Decompositions of Switching Functions," *PSU Report*, Version IV, December 1995.
11. 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*, pp. 13-18, May 1997.
12. M.A. Perkowski, M. Marek-Sadowska, L. Jozwiak, P. Burkey, and S. Mohamed, "Noisy Multiple-Valued Relations: Characterization, Minimization and Decomposition," *PSU EE Dept. Report,* 1997.
13. J.R. Quinlan, "C4.5: Programs for machine learning," San Mateo, CA: *Morgann Kaufmann.*
14. T.D. Ross, M.J. Noviskey, T.N. Taylor, D.A. Gadd, "Pattern Theory: An Engineering Paradigm for Algorithm Design," *Final Technical Report WL-TR-91-1060*, Wright Laboratories, USAF, WL/AART/WPAFB, OH 45433-6543, August 1991.
15. R. Rudell, and A. Sangiovanni-Vincentelli, "Exact Minimization of Multiple-Valued Functions for PLA Optimization," *Proc. ICCAD'86*, pp. 352-355, November 1986.
16. B. Zupan, M. Bohanec, "Learning Concept Hierarchies from Examples by Function Decomposition," Techn. Rep., Dept. Intell. Syst., Jozef Stefan Inst., Ljubljana, Slovenia, Sept. 1996.