BI-DECOMPOSITIONS OF MULTI-VALUED FUNCTIONS FOR CIRCUIT DESIGN AND DATA MINING APPLICATIONS

Bernd Steinbach, Marek A. Perkowski +, and Christian Lang,

Dept. of Computer Science, Freiberg University of Mining and Technology,

Bernhard von Cotta Strasse 1, 09596 Freiberg, Sachsen, Germany, steinb@informatik.tu-freiberg.de

+ Dept. of Electrical and Computer Engineering, Portland State University,

P.O Box 751, Portland, OR 97207-0751, USA, mperkows@ee.pdx.edu

Abstract

We present efficient algorithms for the bi-decomposition of arbitrary incompletely specified functions in variable-valued logic. Several special cases are discussed. The algorithms are especially applicable for Data Mining applications, because, in contrast to the general multi-valued approaches to function decomposition that decompose to arbitrary tables, we create a network from multi-valued two-input operators that are selected by the user. Such decompositions lead to decision rules that are easier to understand by humans.

1 Introduction

Simple Functional Decompositions.
Simple disjoint decompositions $F = H(G(B), A)$ have been used for FPGA synthesis [13], synthesis for layout-driven logic synthesis, Machine Learning and Data Mining [9, 10, 11, 6, 18, 19]. Larger block F of logic is split to smaller blocks G and H, and next blocks G and/or H are recursively split to smaller and smaller blocks, until they become non-decomposable, or until they can be directly realized with some other means (such as in a single PLA block of a CPLD). Set **B** of variables is called the *bound set*, and set **A** is called the set of *free variables.* In binary decomposition the output of function G is encoded as a binary vector of functions $G_i(B)$. In multi-valued decomposition the output of function G is a multi-valued variable. In disjoint decomposition sets **A** and **B** have no common elements. Efficient methods to represent functions in decomposition have been recently created: BDD-Encoded Labeled Rough Partitions [6], and BDD-Encoded Multi-Valued Decision Diagrams [5], but, for the simplification of presentation, in this paper we will use Kmaps to illustrate our concepts. The decomposition principle is always to find the solution with the smallest possible complexity. The complexity can correspond to the total size of non-decomposable blocks, the number of blocks, or some other evaluation of the result. This is obvious for FPGA or layout circuit applications, but it holds also for Machine Learning and Data Mining, where a simpler expression, satisfying the Occam Razor Principle, creates a more meaningful solution, and one that minimizes the learning error [4, 14]. Various decompositional approaches, mainly based on Ashenhurst and Curtis decompositions, have been realized in several programs that can handle both binary and multi-valued, completely and incompletely specified functions and relations, [11]. The methods have been also extended to non-disjoint decompositions [9, 10]. Simple non-disjoint decomposition is: $F = H(G(B_1 \cup C), A_1 \cup C)$, where **C** is the set of *shared variables*. Thus, sets **A** $= A_1\cup$ **C** and **B** $= B_1\cup$ **C** are *non-disjoint.*

Decomposition in Multi-Valued Circuit Design.
Turning our attention to multi-valued circuit design, let us first observe that there exist currently very few methods for synthesis of multi-level multi-valued circuits. They include *factorization methods*, *decision diagram based-methods*, and *decomposition* [3, 11]. Recent decomposers can handle data with don't cares (and multi-valued relations) better than other methods. Two approaches to multi-level multi-valued decomposition are possible. The **first approach**, *Fixed-Multiplicity Decomposition*, [3], assumes that all input and output variables, as well as all intermediate variables G_i that are created in the decomposition, have not more than a fixed constant number of values. For instance, they have at most 3 values for decomposition of ternary functions. This number is called the *multiplicity index.* The advantage of such an approach is that it allows for easier conversion of the tables describing non-decomposable blocks to Multi-Valued Sum of Products (SOP) circuits with some existing MV gates from cell libraries, or with PLA-like function generators. The **second approach**, *Unrestricted-Multiplicity Decomposition*, [11], does not assume any constraints on the number of values, and is thus better suited for Data Mining, where the data are naturally of this type because the attributes (input variables) have various sets of values. For instance, variable SEX can have as few as two values, and variable AGE can have as many as 100 values. After the decomposition of this type to tables with variables of

different radii, the symbolic variables are next encoded with k-valued signals. The advantage of this approach is sometimes finding a decomposition of a smaller cost, because of a more general decomposition model used. Also, this approach allows to find decompositions that are more general than the Curtis decomposition, but are based on very similar principles [3, 9, 10]. We allow there decompositions with **arbitrary** values of multiplicity indices, which is in contrast to Curtis decomposition that in binary case requires the number μ_o of output signals to be smaller than the number μ_i of input signals to the block. In case of MV decomposition this constraint translates to **Ashenhurst decomposition** with single r-valued signal for the block or to a **Curtis-like MV decomposition** with r-valued signals from the block and less output signals than input signals to the block. Our decompositions (both binary and multi-valued) assume none of these constraints. The disadvantage of both Fixed-Multiplicity and Free-Multiplicity decompositional approaches with respect to the MV factorization and MV decision diagram approaches is that arbitrary MV blocks are created as a result of the decomposition. They are specified by tables with k-valued input and k-valued output variables. Such tables require that in order to realize the circuit corresponding to this decomposition, the blocks specified by the tables should be next either further optimized using some other MV synthesis tools, or realized as the not necessarily optimized mv-SOP circuits (using MIN, MAX and literals), that directly correspond to the non-decomposable blocks. From the point of view of circuit realizability of blocks, it would be better to decompose the function to only few selected types of blocks (MV gates), described by multi-valued operators that are directly realizable in hardware. For instance, it can be shown that every function is decomposable to *MV inverters* or *window functions*, and two-input MIN (minimum) blocks. Such realizations, however, may be very far from the minimum. (The ternary MV inverters add modulo-3 constant 1 or 2 to the value. The window literal is defined as follows: $X^i = 2$ for $X{=}i$ and 0 otherwise. These types of single-input functions can be easily extended to any number of values and realized in hardware).

Bi-Decompositions.

Our goal here is to present decompositional approaches that **decompose to finite number of selected block types**, and we will achieve this by the generalization of the bi-decomposition methods introduced originally by Davio and next discussed by many authors: Bochmann [20, 21], Le [22], Steinbach and co-authors [26]-citesteinbach-end, Zakrevskij [36, 37, 38], and Sasao/Butler [12]. The bi-decomposition approach for Boolean logic is based on AND, OR and EXOR

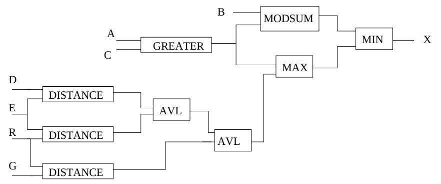

Figure 1. *Netlist with disjoint and non-disjoint bi-decompositions*

decompositions. It decomposes hierarchically an arbitrary function to three types of gates: AND, OR and EXOR. Each type of one-level decomposition, **AND-decomposition, OR-decomposition** or **EXOR-decomposition** can be disjoint or non-disjoint. Every function was proved to be decomposable, when non-disjoint decompositions are also allowed [26]. Observe, that generalizing the bi-decomposition approach to multi-valued logic would allow it to become a **practical tool for multi-level multi-valued design.** Assuming non-disjoint decompositions and availability of negation or window functions plus multi-valued constants, only MIN decompositions would be sufficient (this can be proved by generalizing the proof that NAND and constants is a complete system for binary logic).

In another variant, one would use additionally the MAX gate, which would lead to an MV counterpart of AND/OR/NOT binary logic. Creating such a system is possible and is already included in the approach presented below. The advantage of such approach would be a requirement of having very few standard MV cells. However, it can be observed that the desirable possibility of finding a decomposition with no shared variables or with a small set of shared variables greatly increases with a larger set of operators. Larger set is also a good assumption for Data Mining applications, where many different kinds of relational, mv-logic, arithmetic, and language-based operators are used by knowledge engineers and human experts that create hierarchical rule-based expert systems [17]. The question thus remains, how many gate types would be needed to obtain satisfactory solutions, because too large a number of them would be not practical. Obviously, there should be operators realizing known operators MIN, MAX, MODSUM, AVERAGE, MODULO MULTIPLICATION, TRUNCATED SUM and other. Example of a netlist being a Directed Acyclic Graph of non-disjoint and disjoint single-level bi-decompositions with operators MIN, MODSUM, MAX, GREATER, AVERAGE LOW and DISTANCE is shown in Figure 1.

It was proven by Shannon [15] that when the number of input variables n grows, the ratio of disjointly decom-

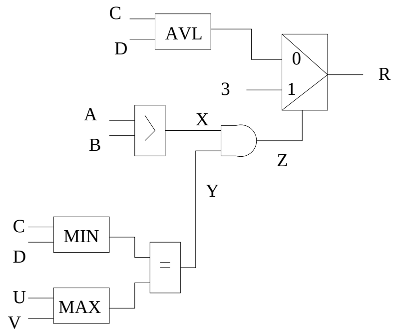

Figure 2. *Netlist from variable-valued operators corresponding to the mixed binary-MV decision rule IF* $(A < B)$ *AND* $MIN(C, D) = MAX(U, V)$ *THEN* $R = 3$ *ELSE* $R = AVERAGE(C, D)$. *Intermediate variables X, Y and Z are binary. Block of MV multiplexer is internally built from non-disjoint decomposition to MAX, two MIN, and two level-changing inverters/buffers.*

posable functions decreases to zero. From the practical point of view, however, this result is not particularly restrictive, because:

(1) Real-life functions are usually decomposable, in contrast to randomly generated worst case functions used in mathematical proofs as one from [15]. For instance, for binary functions it was shown experimentally on large benchmark sets and on many types of real-life benchmarks from different application areas that 82% of them were decomposable, while only 1% of random functions were decomposable [14].

(2) We consider non-disjoint decompositions, for which it was proven that every function is decomposable, and practically with only **few repeated variables** (disjoint decompositions are still preferable). If disjoint decompositions are not possible, we select small sets of shared variables.

(3) Functions with don't cares are better decomposable: the more don't cares, the more probable to find a disjoint decomposition. It is well-known that Data Mining functions have extremely high percent of don't cares. As demonstrated in [11], introducing of shared (the same as repeated) variables has the same effect as dealing with incomplete functions; one repeated binary variable causes creating a new Kmap with half of don't cares.

Bi-Decompositions for Data Mining.

Let us now turn our attention to a non-disjont, operator-based, hierarchical decomposition as a new approach to Data Mining. All Machine Learning and Data Mining methods assume Occam Razor principle, which is practically realized by minimizing **global cost functions** such as Decomposed Function Cardinality (DFC) [11], total multiplicity index (Column Cardinality), or other measures [16]. It was found that there is a very high correlation between the DFC value and the learning error for most of the benchmarks [14, 4]. Another important factor taken into account when evaluating various decomposition structures with blocks is also the *"understandability"* or *"explainability"* property of the resultant set of rules (expression, network, circuit) [17, 16]. Thus, between two sets of rules created by a decomposer that have the same DFC cost, the set of rules that is easier to understand by a human expert is better. Often, the human can further improve the rules, and it was shown that systems that combine computer and human knowledge acquisition on these tasks are better than computers or humans alone [17, 16]. Moreover, expert system users, such as medical doctors or lawyers, do not want to use systems that do not provide explanations that would be comprehensible to them. As one experienced data knowledge engineer working with human experts observed: *"I have never met a doctor who would use EITHER_OR (EXOR) conjunctive in his reasoning"*. Thus, the medical doctor will be never satisfied with the diagnosis provided by an expert system based on a totally "black-box" approach (such as an artificial neural net), or even using complex rules with exotic to them operators such as EXOR or MODSUM. In addition, it was observed that in real-life expert systems the functions realized in blocks are in most cases *monotonic* [17]. Thus monotonic operators should be preferably used in decomposition, or at least given preference during the decomposition choices. Another preference may be to use *symmetric operators.* Concluding, both in MV circuit design and Data Mining applications, there is a need for a method to synthesize a strongly unspecified multi-valued function with arbitrary numbers of values in variables to the preselected set of simple two-argument operators, leading to solution rules such as one illustrated in Figure 2. The goal of this paper is to introduce an efficient method for finding bi-decompositions of strongly unspecified multiple-valued functions, for selected sets of two-input operators.

2 Patterns and Decompositions of Completely Specified Functions

Operators in Ternary Logic.

For simplicity, we will illustrate our approach using ternary logic, but all algorithms can be formulated for arbitrary variable-valued logic using Multivalued Decision Diagrams [5]. Some subset of well-known operators in ternary logic are shown in Figure 3. There are several subsets of them that are functionally com-

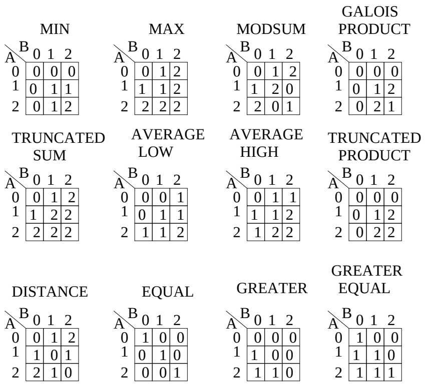

Figure 3. *Selected Operators for Ternary Logic*

plete, with single-input window functions, inversion functions, or with arbitrary universal window functions or literals applied only in the input level (as in MV SOP realizations). Selection of functionally complete subsets is not a topic of this paper because for Data Mining larger sets of operators are better. Operators from Figure 3 are only examples, and the method shown below will lead to solutions, possibly non-optimal, for **any functionally complete set of operators.** It can be easily proven from the fundamental theorem of MV decomposition [11] that if decompositions $F = \phi(G(B), A)$ and $F = \phi(B, H(A))$ exist with multiplicity indices μ_1 and μ_2, respectively, then decomposition $F = \phi(G(B), H(A))$ exists where gate ϕ has its respective inputs with μ_1 and μ_2 values. Thus, there exists a Fixed-Multiplicity Decomposition with $MAX(\mu_1, \mu_2)$ values. For instance, if $\mu_1 = 2$ and $\mu_2 = 3$ then a ternary decomposition of a ternary function F exists.

MIN Decomposition.

To introduce our ideas, we will start with MIN Decompositions of a completely specified function. MIN decomposition has the form:

$f(X) = MIN(g(A), h(B))$.

For instance,

$f(A, B, C, D) = MIN(g(A, B),\ h(C, D))$.

Given is function $f_1(A, B, C, D)$ from Table 1. It can be observed in Table 1 that there are three row patterns and three column patterns. The patterns of rows are:

000 000 000
011 110 101
012 120 201

We say that all rows that have the same pattern are *compatible*. The patterns of columns are:

000
011
012
012
000

↓ CD \ AB →	00	01	02	10	11	12	20	21	22
00	0	0	0	0	0	0	0	0	0
01	0	1	1	1	1	0	1	0	1
02	0	1	2	1	2	0	2	0	1
10	0	1	2	1	2	0	2	0	1
11	0	0	0	0	0	0	0	0	0
12	0	1	1	1	1	0	1	0	1
20	0	1	1	1	1	0	1	0	1
21	0	1	2	1	2	0	2	0	1
22	0	0	0	0	0	0	0	0	0

Table 1. Table for MIN decomposition of ternary function f_1

↓ CD \ AB →	00	01	02	10	11	12	20	21	22	$h(C, D)$
00	0	0	0	0	0	0	0	0	0	0
01	0	1	1	1	1	0	1	0	1	1
02	0	1	2	1	2	0	2	0	1	2
10	0	1	2	1	2	0	2	0	1	2
11	0	0	0	0	0	0	0	0	0	0
12	0	1	1	1	1	0	1	0	1	1
20	0	1	1	1	1	0	1	0	1	1
21	0	1	2	1	2	0	2	0	1	2
22	0	0	0	0	0	0	0	0	0	0
	0	1	2	1	2	0	2	0	1	$g(A, B)$

Table 2. First stage of decomposition. Functions $h(C, D)$ and $g(A, B)$ for MIN decomposition of ternary function f_1. These functions are created by labeling with symbols 0,1 and 2 identical rows and identical columns of the table, respectively

011
011
012
000

Columns with the same pattern are called the *compatible columns.* Table 2 presents the first stage of finding the MIN decomposition. By labeling all (compatible) rows of zeros by 0, all rows of zeros and ones by 1, and all rows of zeros, ones and 2's by 2, we obtain the additional column which describes function $h(C, D)$. By labeling all (compatible) columns of zeros by 0, all (compatible) columns of zeros and ones by 1, and all (compatible) columns of zeros, ones and 2's by 2, we obtain the additional row which describes function $g(A, B)$. From the additional row we can create directly the Kmap of function $g(A, B)$, Figure 4a. By using the permutation operation $(1-2)$ of exchanging rows 1 and 2 of variable C, Figure 4b, we obtain the map of function $h(C, D)$. This is additionally explained in Kmaps from Figure 4c-f. Finally, the entire decomposition of function f_1 can be drawn, Figure 4g. Observe that the inverter (the **1-2 permuter**) operator in variable C was used. There exist three such permuter operators (1-2, 0-1, and 0-2) for ternary logic.

Concluding, the algorithm to find ternary MIN decomposition can be summarized as follows.

[1.] Find all row patterns.

If there are more than 3 patterns, exit.

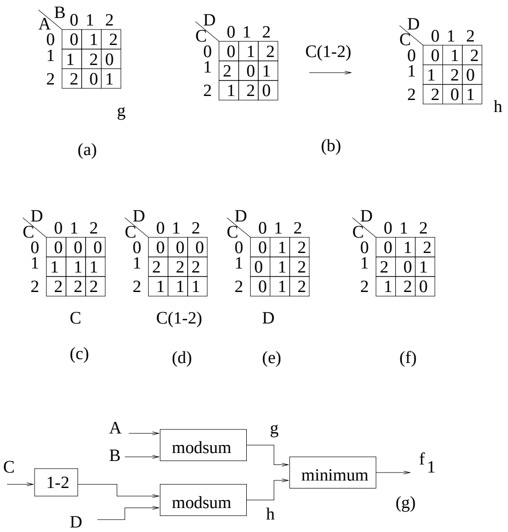

Figure 4. *Second stage of MIN decomposition of function f_1*

[2.] Find all column patterns.
If there are more than 3 patterns, exit.
[3.] Label patterns as presented above. If this is a table of MIN operator (as in Table 3), then MIN decomposition exists.

This method is a generalization of the method from [12]. Let us now explain **another possible approach** to the same problem, this approach generalizes the method from [26]. Analyze the patterns of rows column-by-column. We find that after removing all repeated columns, there are only three patterns: 000, 012 and 011. Similarly, removing all repeated patterns of rows from patterns of columns above, the remaining patterns are: 000, 012 and 011. The same patterns as before. Thus the decomposition operator from Table 3 was found which is the operator of MIN. Both the above methods will become useful, when it comes to find patterns in cofactors of large bound and free sets for incompletely specified functions.

MAX Decomposition.
Given is function f_2 from Table 4, with bound and free sets of variables $\mathbf{B} = \{\text{A,B}\}$ and $\mathbf{A} = \{\text{C,D}\}$. Following the first procedure above, Table 5, we find the realization from Figure 5 with the same functions h and g as previously. Now, following the second procedure above, but labeling the columns and rows of 0's, 1's and 2's as 0; the columns and rows of 1's and 2's as 1; and the columns and rows of 2's as 2; we found that the reduced pattern table, Table 6, is the table of operator MAX. Analogously, the same solution is found using the second approach which we leave to the Reader.

MODSUM Decomposition.

$g \setminus h$	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

Table 3. Table of ternary MIN operator for function f_1

↓ CD \ AB →	00	01	02	10	11	12	20	21	22
00	0	1	2	1	2	0	2	0	1
01	1	1	2	1	2	1	2	1	1
02	2	2	2	2	2	2	2	2	2
10	2	2	2	2	2	2	2	2	2
11	0	1	2	1	2	0	2	0	1
12	1	1	2	1	2	1	2	1	1
20	1	1	2	1	2	1	2	1	1
21	2	2	2	2	2	2	2	2	2
22	0	1	2	1	2	0	2	0	1

Table 4. Table for MAX decomposition of ternary function f_2

↓ CD \ AB →	00	01	02	10	11	12	20	21	22	$h(C,D)$
00	0	1	2	1	2	0	2	0	1	0
01	1	1	2	1	2	1	2	1	1	1
02	2	2	2	2	2	2	2	2	2	2
10	2	2	2	2	2	2	2	2	2	2
11	0	1	2	1	2	0	2	0	1	0
12	1	1	2	1	2	1	2	1	1	1
20	1	1	2	1	2	1	2	1	1	1
21	2	2	2	2	2	2	2	2	2	2
22	0	1	2	1	2	0	2	0	1	0
	0	1	2	1	2	0	2	0	1	$g(A,B)$

Table 5. Table for functions $h(C,D)$ and $g(A,B)$ for MAX decomposition of ternary function f_2. These functions were calculated analogously as in Table 2

	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

Table 6. Table of MAX operator

↓ CD \ AB →	00	01	02	10	11	12	20	21	22
00	0	1	2	1	2	0	2	0	1
01	1	2	0	2	0	1	0	1	2
02	2	0	1	0	1	2	1	2	0
10	2	0	1	0	1	2	1	2	0
11	0	1	2	1	2	0	2	0	1
12	1	2	0	2	0	1	0	1	2
20	1	2	0	2	0	1	0	1	2
21	2	0	1	0	1	2	1	2	0
22	0	1	2	1	2	0	2	0	1

Table 7. Table for MODSUM decomposition of ternary function f_3

	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

Table 8. Table of ternary MODSUM operator

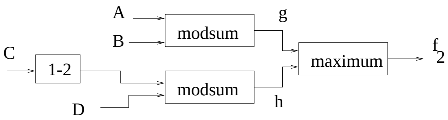

Figure 5. *MAX decomposition for function f_2*

↓ CD \ AB →	00	01	02	10	11	12	20	21	22
00	0	0	0	0	0	0	0	0	0
01	1	1	1	1	1	1	1	1	1
02	2	2	2	2	2	2	2	2	2
10	2	2	2	2	2	2	2	2	2
11	0	0	0	0	0	0	0	0	0
12	1	1	1	1	1	1	1	1	1
20	1	1	1	1	1	1	1	1	1
21	2	2	2	2	2	2	2	2	2
22	0	0	0	0	0	0	0	0	0

Table 9. Table of function $temp$ for MODSUM Decomposition of function f_3

Given is function f_3 from Table 7, with bound set of variables $\mathbf{B}$ = {A,B} and free set of variables $\mathbf{A}$ = {C,D}. Following the same two procedures as previously, we find that decomposition $f_3 = MODSUM(g(A,B),\ h(C,D))$, with the same functions $h(C,D)$ and $g(A,B)$ as in the two previous examples, and the reduced pattern table, Table 8 is the table of operator MODSUM.

In this case, **one more approach** is possible:

(1) calculate function g: first row of f_3: 012120201 g.

(2) calculate a temporary function
$temp = Mod\_diff(f_3(A,B,C,D),\ g(A,B))$.

(3) calculate function h: first column of f (constant values in each row are neccessary for MODSUM decomposition)

CD	h
00	0
01	1
02	2
10	2
11	0
12	1
20	1
21	2
22	0

Functions g and h can be found as previously and the operator table of the decomposition operator is MODSUM, see Table 8.

Composing two-input operators from other operators.

Even if a decomposition with an unknown operator is found using the method of compatible columns, this operator can be still composed from the known operators.

Example: Function f_4 from table in Figure 6 cannot be decomposed by MIN(A,B), MAX(A,B) or MODSUM(A,B). But $f_4 = MAX(g,h)$, where functions g and h are as in Figure 6b,c. Thus, $f_4 = MAX(MODSUM(011,002), MODSUM(002,011))$ see Figure 6d, where $<011>$ and $<002>$ are universal literals.

Conclusion on these examples.

The above three examples suggest two general procedures for a completely specified function. The **first procedure**, that we will call the *Compatibility Algorithm*, is based on labeling arbitrary compatible patterns, calculating multiplicity indices, and next finding tables of functions F, g and h. This procedure is the same for any two-input decomposing operator, it does not assume any specific operator, and the operator becomes known only when the decomposition is completed. Thus, the user has no possibility to influence the decomposition type. This may lead to an operator that is not one of the selected operators. On the other hand, this method allows for permuting inputs, thus each operator pattern corresponds to one of many possible compositions of two-input and single input operators, so it is efficient. In case of incompletely specified function, the Compatibility Algorithm would require, however, coloring of two large *incompatibility graphs*; one with nodes for the cofactors of the bound set, and the other one with nodes for the cofactors of the free set [10]. The graphs are large, because we split the set of input variables into two sets of approximately the same size. If for both of these graphs we are able to find coloring with 3 or less colors (multiplicity index 3 or less), the binate decomposition exists. In contrast to the 2-Colorability, which is polynomial and for which we have implemented an efficient algorithm [8], the 3-Colorability Problem is NP-complete. To complicate things more, because the minimal graph coloring is usually not unique, several decompositions can be found, with various decomposition operators. Such coloring-based procedure would be not very efficient, and would not give preferences to the desired decomposition types such as monotonic, or symmetric. The **second approach**, called *Operator Pattern Algorithm*, assumes certain type(s) of decomposing operator, and looks for a decomposition only with respect to this operator. It is thus an *operator-oriented decomposition.* Therefore below, for incomplete functions, we will present a more efficient and comprehensive approach, that will make use of specific properties of the bi-decomposition of multi-

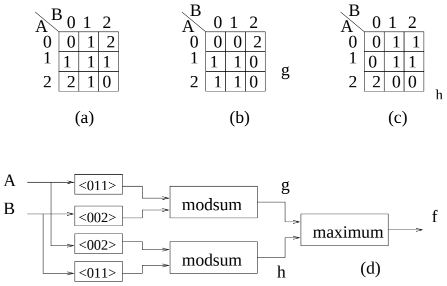

Figure 6. *Schematic of composed realization of an operator using MAX decomposition for function f_4*

valued incomplete functions, and will combine the ideas from both the algorithms.

3 Decompositions of Incompletely Specified Functions

Number of Decompositions.
In case of a k-valued logic and n variables, there are k^{k^n} functions. If we combine two of these functions, one for rows and one for columns, we get $k^{k^n} * k^{k^n} = k^{k^n + k^n}$ functions realizable by decomposition. This number is much smaller than the number of all possible functions of $2 * n$ variables: $k^{k^{2n}}$ functions. Taking thus an arbitrary randomly selected function of n variables, we have a probability of $1 - \frac{k^{2*k^n}}{k^{k^{2n}}}$ that this function is **not** decomposable. Thus random complete functions of many variables are very unlikely to be disjoint-decomposable. Recall, however, that an incomplete function of $2n$ variables specified on K cares corresponds to $k^{k^{2n}-K}$ complete functions. Thus taking an arbitrary function with K cares, we have the probability of $(1 - \frac{k^{2*k^n}}{k^{k^{2n}}})^{k^{k^{2n}-K}}$ that this function is not decomposable. When K/k^{2n} is small, as in Data Mining where it can be less than 0.01 percent, the probability of finding disjoint decomposition is then quite high, even for random functions. Still, if the disjoint decomposition does not exist, addition of every shared variable improves the chance of finding a decomposition. The procedure, however, may become inefficient, so good methods for finding bound and free sets are needed, as well as a fast algorithm for basic decomposition execution step (because it is repeated very many times).

MIN, MAX and Simple Monotonic Decompositions.
MIN Decomposition of an incompletely specified function is based on the same principles as illustrated in previous sections for complete functions. Simplified algorithm for MIN Decomposition is the following:

[1] Find rows and columns with pattern of only 0's and don't cares, label respective rows and columns by 0. Remove 0's from these rows and columns.

[2] Find rows and columns with compatible patterns of 1's and don't cares, label respective rows and columns by 1. Remove 1's from these rows and columns. If there is more than one compatible pattern of 0's and 1's, exit.

[3] Find rows with compatible patterns of 2's and don't cares, label respective rows and columns by 2. Remove 2's from these rows and columns. If some patterns remain, exit.

[4] The labeled rows and columns determine the MIN operator pattern.

To understand this algorithm the Reader is advised to apply it to our previous tables from MIN decomposition. Also, try to apply it to the MIN table from Figure 3. Observe that similar algorithm can be created for MAX decomposition, but the order of removals will be not [0,1,2] as for MIN presented above, but [2,1,0]. This can be checked on the MAX table from Figure 3. Similarly it can be checked in Figure 3 that TRUNCATED-PRODUCT operator (TRUNCATED-PRODUCT decomposition) will be found for order [0,2,1], GREATER operator for order [0,1,0], and GREATER-EQUAL operator for order [1,0,1]. All such functions, for which the decomposition can be checked by the parametrized variants of the above algorithm with different orders of removals, we will call the *Simple Monotonic Functions*. Functions for which the removal can be done partially are also good candidates for non-disjoint decompositions, and we call them the *Partially Monotonic Functions*. In tables from Figure 3, GALOIS-PRODUCT and TRUNCATED-SUM are such functions.

Preprocessing for graph coloring.
Because graph coloring [8, 9, 11] can be slow, we propose to apply the following algorithm that in many cases removes totally the necessity of coloring, and in other cases reduces significantly the size of the rows and column incompatibility graphs.

[1] Combine all rows that have the same overlapping symbols.

[2] Combine all columns that have the same overlapping symbols.

[3] Iterate until no rows or columns can be combined.

[4] If the size of the resultant matrix is 3 x 3, bi-decomposition is found. The decomposition operator is specified by the matrix. This matrix can have don't cares, thus specifying various decomposition operators. Moreover, various functions g and h can be created from the traces of the combined (i.e., labeled the same way) rows or columns in the process of combining. If more than 3 rows or columns were found that cannot be combined to 3 patterns, exit.

[5] Execute two Graph Colorings for functions g and h. For each, if the chromatic number is larger than 3, exit. Otherwise combine in each the nodes colored with the same color creating the operator table.

Example. Given is the table of function $f_6(A, B, C, D)$ from Table 10. By combining rows table 11 is created, and next by combining columns in it, Table 12. Observe that by combining first columns and next rows, Tables 13, 14, another solution is found. Observe in Table 14 the final matrix specifying the ternary decomposition operator. Note, the decomposition is different than in Table 12 because row labels are different. Because of a don't care in the operator table, it corresponds to three different decomposition operators; with the don't care replaced with 0, with 1, or with 2. Thus, several solutions may be found, and the one corresponding to the realizable decomposition operator is next chosen.

4 Experimental Results and Conclusions

Table 15 gives results of bi-decomposition of some Benchmark functions from POLO directory [4, 11, 16] (Data Mining and Machine Learning). We tried decomposition for the MIN and MAX operator by successive removal of values as described in section 3 for simple monotonic functions. That is, function $f(xa, xb, xc) = MIN/MAX(g(xa, xc), h(xb, xc))$. The meaning of the columns is: $Name$ = Name of the Benchmark, $\#In$ = Number of multi-valued inputs, $log(In) = log_2$(Product of multiplicity indices of inputs), Out = multiplicity index of output, $\#GN$ = number of (multi-valued) inputs of g for MIN, $\#HN$ = number of (multi-valued) inputs of h for MIN, $\#GX$ = number of (multi-valued) inputs of g for MAX, $\#HX$ = number of (multi-valued) inputs of h for MAX. The dashes mean that there is no decomposition. The total computation time was 10 minutes for all Benchmarks of the table on a 133 MHz Pentium Processor.

We generalized the binary bi-decompositions discussed previously by Steinbach and Sasao/Butler for the case of **arbitrary variable-valued logic**. Although bi-decomposition is only a special case that can be derived from the general-purpose Ashenhurst/Curtis decomposition, this decomposition is especially important in Knowledge Discovery, Data Mining, and automatic knowledge acquisition applications, because **it creates networks that when converted to sets of rules, are easier to understand.** Moreover, although every function is disjointly or non-disjointly decomposable this way, the MV bi-decompositions are especially efficient and effective for the case of **strongly unspecified** functions, characteristic for Data Mining.

References

[1] R. L. Ashenhurst, "The decomposition of switching functions," *Proc. Int. Symp. Th. Swi.*, pp. 74-116, April 1957.

[2] H. A. Curtis, "A New Approach to the Design of Switching Circuits," Princeton, N.J.: *Van Nostrand*, 1962.

[3] C. Files, R. Drechsler, and M. Perkowski, "Functional Decomposition of MVL Functions using Multi-Valued Decision Diagrams," *Proc. ISMVL'97*, pp. 27-32, 1997.

[4] C. Files, and M. Perkowski, "An Error Reducing Approach to Machine Learning using Multi-Valued Functional Decomposition," *Proc. ISMVL'97*, pp. 27-32, 1998.

[5] C. Files and M. Perkowski, "Implementing Multi-Valued Decision Diagram Package using Binary Decision Diagrams," *submitted.*

[6] S. Grygiel, M. Perkowski, M. Marek-Sadowska, T. Luba, and L. Jozwiak, "Cube Diagram Bundles, A New Representation of Strongly Unspecified Multiple-Valued Functions and Relations," *Proc. ISMVL'97,* May 1997, pp. 287 - 292. http://www.ee.pdx.edu/ mperkows/ML/=xxx.ps

$\downarrow$ CD \ AB $\rightarrow$	00	01	02	10	11	12	$h(C, D)$ label
00	0	-	0	-	1	-	a
01	0	-	-	1	1	-	b
02	-	-	0	-	1	-	c
10	0	-	0	-	-	-	d
11	0	0	-	-	-	-	e
12	0	-	-	-	-	0	f
20	0	2	2	1	1	0	g
21	0	2	2	-	1	0	h
22	-	-	2	-	1	-	i
	k	l	m	n	o	p	$g(A, B)$ label

Table 10. Table for incomplete function f_6 with labeled rows and columns

						row labels $\downarrow$
0	-	0	1	1	-	a,b,c
0	0	0	-	-	0	d,e,f
0	2	2	1	1	0	g,h,i
k	l	m	n	o	p	$\leftarrow$ column labels

Table 11. Combined rows for function from Table 10

			row labels $\downarrow$
0	0	1	a,b,c
0	0	-	d,e,f
0	2	1	g,h,i
k,p	l,m	n,o	$\leftarrow$ column labels

Table 12. Combined columns for function from Table 11, the final matrix specifying the ternary decomposition operator

			row labels $\downarrow$
0	0	1	a
0	-	1	b
-	0	1	c
0	0	-	d
0	0	-	e
0	-	-	f
0	2	1	g
0	2	1	h
-	2	1	i
k,p	l,m	n,o	$\leftarrow$ column labels

Table 13. Combined columns for function from Table 10

			row labels $\downarrow$
0	0	1	a,c
0	0	-	d,e
0	2	1	b,f,g,h,i
k,p	l,m	n,o	$\leftarrow$ column labels

Table 14. Combined rows for function from Table 13

Name	#In	log(In)	Out	#GN	#HN	#GX	#HX
lensesmv.ml	4	5	3	3	3	3	1
shuttlem.ml	6	8	2	4	4	5	1
ships.ml	4	8	4	3	1	3	1
hayes.ml	4	9	3	-	-	-	-
iris.ml	4	12	3	3	2	3	3
post-operative.ml	8	12	3	6	5	6	5
cloud.ml	6	18	2	4	3	4	3
monks3tr.ml	6	9	2	4	2	4	2
monks1tr.ml	6	9	2	3	3	3	3
bridges1.ml	9	15	7	7	4	7	6
monks2tr.ml	6	9	2	-	-	-	-
bridges2.ml	10	17	7	7	7	7	5
zoo.ml	16	19	7	8	8	8	8
irish.ml	4	9	3	2	2	2	2
monks1te.ml	6	9	2	3	3	3	3
monks2te.ml	6	9	2	3	3	3	3
monks3te.ml	6	9	2	3	3	3	3
balance.ml	4	10	3	-	-	-	-
sensory.ml	11	18	11	6	5	6	5
house-votes-84.ml	16	16	2	14	14	-	-
tic-tac-toe.ml	9	15	2	7	7	7	7
flare1.ml	10	16	2	5	5	6	5
car.ml	6	11	4	-	-	-	-
flare2.ml	10	16	3	5	5	5	5
employ1.ml	9	14	4	5	4	-	-
nursery.ml	8	14	5	-	-	-	-
employ2.ml	7	15	4	5	5	-	-
chess1.ml	6	16	18	-	-	-	-

Table 15. Experimental Results

[7] Y-T. Lai, M. Pedram, S. B. K. Vrudhula, "EVBDD-based algorithm for integer linear programming, spectral transformation, and functional decomposition," *IEEE Trans. CAD,* Vol. 13, No. 8. Aug. 1994, pp. 959-975.

[8] M. A. Perkowski, "A New Representation of Strongly Unspecified Switching Functions and Its Application to Multi-Level AND/OR/EXOR Synthesis," *Proc. RM'95*, 27-29 Aug. 1995, pp. 143-151.

[9] M.A. Perkowski, S. Grygiel, and the Functional Decomposition Group, Department of Electrical Engineering, "A Survey of Literature on Function Decomposition," Version IV, *PSU ECE Dept. Report,* Nov. 20, 1995.

[10] M.A. Perkowski, T. Luba, S. Grygiel, P. Burkey, M. Burns, N. Iliev, M. Kolsteren, R. Lisanke, R. Malvi, Z. Wang, H. Wu, F. Yang, S. Zhou, and J.S. Zhang, " Unified Approach to Functional Decompositions of Switching Functions," *PSU Electr. Engn. Dept. Report,* Dec. 29, 1995.

[11] M. Perkowski, M. Marek-Sadowska, L. Jozwiak, T. Luba, S. Grygiel, M. Nowicka, R. Malvi, Z. Wang, and J. S. Zhang, "Decomposition of Multiple-Valued Relations," *Proc. ISMVL'97,* Halifax, Nova Scotia, Canada, May 1997, pp. 13 - 18. http://www.ee.pdx.edu/ mperkows/ML/=p33.ps.

[12] T. Sasao and J. Butler, "On Bi-Decompositions of Logic Functions," *Proc. Intern. Workshop on Logic Synthesis,* Lake Tahoe, California, May 18-21, 1997.

[13] W. Wan, and M. A. Perkowski, "A New Approach to the Decomposition of Incompletely Specified Functions based on Graph-Coloring and Local Transformations and Its Application to FPGA Mapping," *Proc. EURO-DAC '92*, pp. 230 - 235, Sept. 7-10, Hamburg, 1992. http://www.ee.pdx.edu/ mperkows/ML/pap.ps.

[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] C. E. Shannon, "The synthesis of two-terminal switching circuits," *Bell Syst. Techn. J.*, Vol. 28, 1949, pp. 59-98.

[16] B. Zupan, M. Bohanec, J. Demsar, and I. Bratko, "Feature Transformation by Function Decomposition," *IEEE Expert,* Spec. Issue on Feature Transformation and Subset Selection.

[17] B. Zupan, M. Bohanec, I. Bratko, B. Cestnik, "A Dataset decomposition approach to data mining and machine discovery," *preprint from authors,* 1998.

[18] M. Burns, M. Perkowski, L. Jozwiak, "An Efficient Approach to Decomposition of Multi-Output Boolean Functions with Large Sets of Bound Variables," *Proc. 1998 Euromicro*, pp. 16 - 23.

[19] R. Malvi, M. Perkowski, and L. Jozwiak, "Exact Graph Coloring for Functional Decomposition: Do we Need it?," *Proc. 3rd Int. Work. Boolean Problems*, 1998, pp. 1 - 10.

[20] D. Bochmann, and B. Steinbach, "Logikentwurf mit XBOOLE," *Verlag Technik*, Berlin 1991.

[21] D. Bochmann, F. Dresig, and B. Steinbach, "A New Decomposition Method for Multilevel Circuit Design," *European Design Automation Conference,* Amsterdam, 1991, pp. 374 - 377.

[22] T.Q. Le, and B. Steinbach, "Effiziente Methoden zum Logikentwurf testbarer kombinatorischer Schaltungen und deren impliziten Testsatzberechnung," *Proc. 3rd ITG/GI-Workshop: "Testmethoden und Zuverlassigkeit von Schaltungen und Systemen",* Blomberg, 1991.

[23] B. Steinbach, and T.Q. Le, "Entwurf testbarer Schaltnetzwerke," *Wissenschaftliche Schriftenreihe der TU Chemnitz,* H. 12/1990.

[24] B. Steinbach, "Auflosbarkeit und Eindeutigkeit Boolescher Gleichungen," *Wissenschaftliche Schriftenreihe der TU Chemnitz-Zwickau,* H. 7/1992.

[25] B. Steinbach, "XBOOLE - A Toolbox for Modelling, Simulation, and Analysis of Large Digital Systems," *System Analysis and Modelling Simulation,* Gordon & Breach Science Publishers, 9(1992), No. 4, pp. 297 - 312.

[26] B. Steinbach, F. Schumann, and M. Stoeckert, "Functional Decomposition of Speed Optimized Circuits," *in: Auvergne, D.; Hartenstein, R.: Power and Timing Modelling for Performance of Integrated Circuits,* IT Press Verlag, Bruchsal, 1993, pp. 65 - 77.

[27] B. Steinbach, and M. Stoeckert, "Design of Fully Testable Circuits by Functional Decomposition and Implicit Test Pattern Generation," *Proc. 12th IEEE VLSI Test Symposium*, Cherry Hill, New Jersey, 1994, pp. 22 - 27.

[28] B. Steinbach, and Th. Muller, "Dekompositorische Schaltungssynthese mit komplexen Logikmoduln," *Tagungunterlagen des GI/ITG - Workshops "Anwender- programmierbare Schaltungen",* Karlsruhe, 1994.

[29] B. Steinbach, and A. Wereszczynski, "Synthesis of Multi-Level Circuits Using EXOR-Gates," *Proc. RM'95*, Chiba (Makuhari), Japan, 1995, pp. 161 - 168.

[30] B. Steinbach, and K. Hesse, "Design of large digital circuits utilizing functional and structural properties," *in: Steinbach, B. (Hrsg.): Boolesche Probleme, Proceedings des 2. Workshops,* 19. und 20. September 1996, TU Bergakademie Freiberg, pp. 23 - 30.

[31] B. Steinbach, and Z. Zhang, "Synthesis for Full Testability of Large Partitioned Combinational Circuits," *in: Steinbach, B. (Hrsg.): Boolesche Probleme, Proc. 2nd Workshop,* Sept. 19-20, TU Bergakademie Freiberg, pp. 31 - 38.

[32] B. Steinbach, and Z. Zhang, "Designing for Testability of Long Pipeline of Modules," in: Anheier, W.; (ed): *Tagungsband - 9. Workshop "Testmethoden und Zuverlassigkeit von Schaltungen und Systemen",* Report 1/97, Berichte Elektrotechnik, Universitaet Bremen, 1997, pp. 74 - 77.

[33] B. Steinbach, and Z. Zhang, "Synthesis for Full Testability of Partitioned Combinational Circuits Using Boolean Differential Calculus," *in: Proc. IWLS'97,* Granlibakken Resort - Tahoe City, CA - USA, 1997, pp. 1 - 4.

[34] B. Steinbach, Z. Zhang, and Ch. Lang, "Logical Design of Fully Testable Large Circuits by Decomposition," *in: Proc. Second Intern. Conf. on Computer-Aided Design of Discrete Devices*, (CAD DD'97), Vol. 1, pp. 7 - 14, Minsk, Belarus.(http://www.informatik.tu-freiberg.de/prof2/publikationen/ld_ftc_d.ps).

[35] B. Steinbach, and A. Zakrevskij, "Three Models and Some Theorems on Decomposition of Boolean Function," *in: Steinbach, B. (Hrsg.): Boolean Problems, Proc. 3rd Intern. Workshops on Boolean Problems,* Sept. 17-18, 1998, TU Bergakademie Freiberg, pp. 11 - 18.

[36] A.D. Zakrevskij, "An algorithm for decomposition of Boolean functions," *Trudy SPhTI,* is.44, 1964, Tomsk, pp. 5-16 (in Russian).

[37] A.D. Zakrevskij, "PLA decomposition over input variables," *Doklady AN B,* 1980, Vol. 24, No. 5, pp. 419-422 (in Russian).

[38] A.D. Zakrevskij, "On a special kind decomposition of weakly specified Boolean functions," *Computer-Aided Design of Discrete Devices,* pp. 36–45, Minsk, 1997.