AN EFFICIENT APPROACH TO DECOMPOSITION OF MULTI-OUTPUT BOOLEAN FUNCTIONS WITH LARGE SETS OF BOUND VARIABLES

Michael Burns, Marek Perkowski

Department of Electrical Engineering
Portland State University
Portland, OR 97207, USA

Lech Jozwiak

Faculty of Electronics Engineering
Technical University of Eindhoven
MB 5600 Eindhoven, The Netherlands

Abstract

Finding appropriate bound sets of variables is the most important task of functional decomposition. When solving some problems the bound sets need to be large; for instance in decomposition to symmetric subfunctions realized in MOPS arrays for submicron technologies, or when no good small bound sets exist. In such cases the creation of the incompatibility graph, that is necessary to evaluate good variable partitionings, becomes very inefficient. Therefore an algorithm is proposed here, that can speed up this process by orders of magnitude without sacrifying the quality of the decomposition, because the same graph coloring algorithms (exact or approximate) is still applied to the created graph.

1 Introduction

Modern microelectronic technology has created opportunities to build systems on chip and reconfigurable custom computing machines. However, these opportunities cannot be effectively exploited due to the weakness of the traditional architectures and logic synthesis methods and tools. One of the most promising approaches of modern logic synthesis is general functional decomposition [5, 6, 7, 10]. The most important problem of the functional decomposition is finding sets of input variables for sub-functions that result in high-quality decompositions. For solving this problem the column incompatiblity or compatibility graph is used in the Curtis-style functional decomposition [12, 7, 10]. Column compatibility checking is the process of constructing a compatibility or incompatibility graph. The nodes of the graph represent columns(or groups of columns) of the Kmap and edges represent the compatibility relationship between the columns. The graph is used in the major step of the decomposition process, i.e. the column minimization (for instance, by node coloring of this graph). The graph must be constructed many times for various sets of bound variables in order to find the best set of bound variables; i.e., the set leading to a decomposition that minimizes certain circuit-related cost function, such as the number of gates, total delay, area, etc..

This paper presents a new approach which can significantly decrease the time required for column compatibility checking over the other approaches to Curtis-style decomposition, especially in the cases where large graphs are constructed. Our approach preserves the quality of the decomposition, because **the same data is obtained for column compatibility, but much faster.** Large graphs are often created for symmetry-based decomposition in layout-driven logic synthesis for submicron technologies [11] and some other applications, such as generalized PLA partitioning for CPLDs, FPGAs mapping, and other. Also, some of the highest quality decompositions of standard types may require large graphs, especially, when no good small bound sets exist. Typically, the number of nodes in a compatibility or incompatibility graph grows with the number of variables in the bound set. Unfortunately, little is published about the use of large bound sets in Curtis-style decompositions. This is most likely due to the application of the functional decomposition to the FPGAs with lookup tables of not more than 5 inputs, and the increased computation time required for partitioning, column compatibility checking, column minimization and encoding when bound sets are large.

Previously, column compatibility checking was performed in a pairwise fashion referred to here as the Pair Compatibility Approach(**PCA**) [7, 8, 12, 10, 4]. The basic idea behind the **PCA** is to check the compatibility relationship of each column with every other column one pair at a time. The new approach presented in this paper is referred to as the Group Compatibility Approach(**GCA**). The basic idea behind the **GCA** is to check the compatibility relationship between the pairs of **groups of**

columns instead of checking the compatibility between each pair of **single columns**. Note that the columns and the blocks of the bound set [7] are referred to in this paper simply as columns. Similarly, rows and blocks of the free set are referred to in this paper simply as rows. One reason to do it is that blocks of the bound set and blocks of the free set are treated the same way in the algorithms as rows and columns. The another reason is the graphical explanation of the approaches (it is much simpler to visualize rows and columns in a Kmap than blocks of the free set or bound set).

The organization of this paper is as follows: In Sect. 3, the algorithm for the classical approach(**PCA**) to the column compatibility checking problem is briefly reviewed for comparison. In Sect. 4, the algorithms for the new approach(**GCA**) to the column compatibility checking problem are introduced. Also presented in Sections 3 and 4 is an example illustrating differences between the **PCA** and **GCA** approaches. In Sect. 5 we compare the **PCA** and **GCA** approaches. In Sect. 6 comparisons of results are given, and Sect. 7 concludes the paper.

2 Definitions and Notations

Definition 1. Repeated cubes are care cubes which are elements of more than one output class.
Example:

```
1  0111 00
2  0001 01
3  0110 0-
4  1010 10
5  0010 11
```

There are four output classes corresponding to output vectors 00, 01, 10, and 11. From these vectors, cubes are classified: P(F) = {(1,3);(2,3);(4);(5)}. Since cube number 3 is an element of more than one output class, it is referred to as a repeated cube.
Definition 2. Classes of cubes are groups of cubes that have compatible output values.
Definition 3. Classes of columns are groups of columns which have compatible outputs within some subset of rows of the Kmap. However, unless otherwise specified, this does not imply that the all columns in a class are compatible with each other within all rows of the Kmap.
Definition 4. Incompatible classes are classes which are incompatible with some other class.
Definition 5. Incompatible classes of cubes are classes in which some or all cubes in one class are incompatible with some or all cubes in some other class.
Definition 6. Incompatible classes of columns are classes in which all columns in one class are incompatible with all columns in some other class.

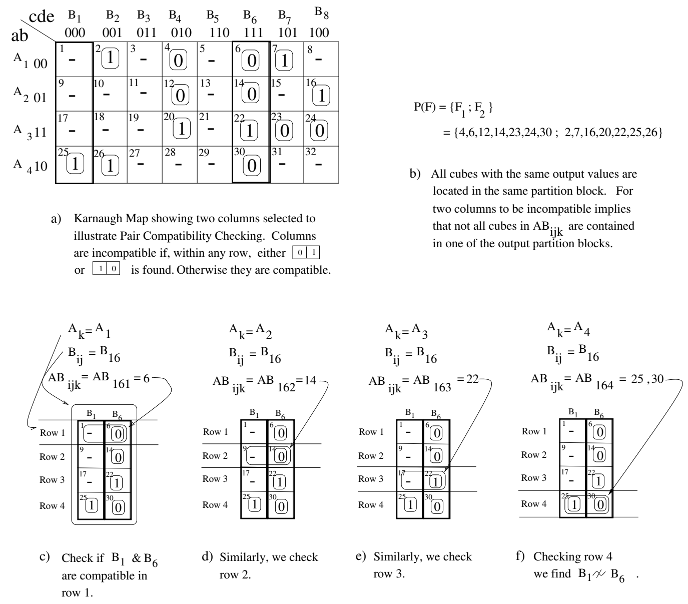


Figure 1: *Diagram illustrating* **PCA** *approach to Column Compatibility Checking for Single Output Function F2.*

Definition 7. Pairs of incompatible classes of cubes are two specific classes of cubes in which some or all cubes in one class are incompatible with some or all cubes in the other class. Unless otherwise specified, the pairs of incompatible classes of cubes are rough partitions [7] of cubes which are elements of the same row but not of the same output class.
Definition 8. Pairs of incompatible classes of columns are two specific classes of columns in which all columns in one class are incompatible with all columns in the other class.
Definition 9. Output classes are the individual partition blocks within the output partition. The cubes which belong to each output class have compatible output values.
$\mathbf{A_i}$ = i_{th} partition block in partition $P(A)$ of the set of cubes. In simpler terms, A_i is a set of cubes corresponding to a row(or rows) in the Kmap. A is a **free set of variables**, free variables corresponding to rows of the Kmap.
$\mathbf{B_i}$ = i_{th} partition block in $P(B)$. In simpler terms, B_i is a set of cubes corresponding to a column(or columns) in the Kmap. B is a **bound set of variables**, bound variables corresponding to columns of the Kmap.
$\mathbf{F_i}$ = i_{th} partition block in $P(F)$. In simpler terms, F_i is a set of cubes which have compatible output values.
$\mathbf{B_{ij}}$ denotes an *edge* between columns B_i and B_j when referring to a compatibility/incompatibility graph.
$\mathbf{AB_{ijk}} = AB_{ijk} = (B_i \cup B_j) \cap A_k$ = The set of all cubes that are elements of blocks B_i or B_j that are also elements of the same block A_k of the free set.
$\mathbf{IC_{ij}}$ denotes the class of cubes from row A_i that are elements of the same output class F_j. Cubes are elements

Shown circled with dashed lines are the classes chosen for the final cover set.

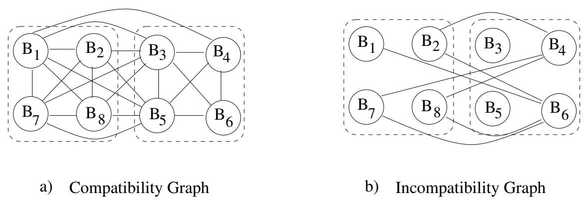


Figure 2: *Compatibility and Incompatibility Graphs for function F2 from Fig.1.*

of the same output class F_j if they have compatible output values. For single output binary functions, one of the classes will have output value 0 while the other will have output value 1.

IC is the set of pairs of incompatible classes of cubes; (IC_{ij}, IC_{ik}). All classes IC_{ij} are incompatible with all classes IC_{ik} for all pairs in row i, and for all rows i.

$\mathbf{IR_{ir}}$ is the class of cubes that are incompatible with repeated cube r in row i.

IR is the set of pairs of incompatible classes of cubes of the form $(r,\ IR_{ir})$.

IB is the set of pairs of incompatible classes of columns of the form (IB_{ij}, IB_{ik}). All classes IB_{ij} are incompatible with all classes IB_{ik} for all pairs.

$\mathbf{IB_{ij}}$ denotes the set of columns(B_n) which have at least one cube in common with the class of cubes IC_{ij}. Expressed in simpler terms, IB_{ij} denotes a set of columns which have the same output values for a particular cofactor(or row) of the Kmap.

$\mathbf{SR_i}$ denotes the set of repeated cubes that are found in row i of the Kmap.

$\mathbf{R_{ir}}$ is the set of cubes which belong to at least one class IC_{ij} which cube r also belongs to. (R_{ir} represents the set of all "care" cubes compatible with cube r in row i).

3 Classical Approach To Column Compatibility Checking: PCA

The following is a brief explanation of the **PCA** approach to column compatibility checking. The goal of this algorithm is to obtain either an incompatibility graph or a compatibility graph. Recall that a compatibility graph is simply the complement of the incompatibility graph. Fig. 1a shows the Kmap used to illustrate the **PCA** algorithm. In general, this algorithm checks the compatibility of each pair of columns and if compatible, then assigns an edge in the compatibility graph between the two nodes corresponding to the two columns. The method used for checking the compatibility of two columns is straightforward and requires little explanation.

To determine if two columns are compatible, the output values of the columns are checked to see if they are compatible for each combination of the free set variables(i.e., each row). The order that each pair of columns are checked for compatibility is arbitrary. Begin by arbitrarily selecting the two columns highlighted in Fig. 1a (columns B_1 and B_6). Fig. 1b shows the output partition $P(F)$ with cubes classified according to their output values. In Fig. 1c-f shown are the compatibility checks within each row necessary to determine if the two columns are compatible. Columns B_1 and B_6 are compatible within rows 1 thru 3, but are incompatible in row 4. Therefore, column B_1 is incompatible with column B_6. The remaining pairs of columns in Fig. 2a are checked in the same manner. This results in the compatibility graph (Fig. 2b) with edges between nodes in the graph indicating that two columns are compatible.

4 New Approach To Column Compatibility Checking: GCA

The new algorithms presented in the following subsections can greatly reduce the number of calculations required to create the compatibility or incompatibility graph when there is a large number of blocks in the bound set.

The basic idea of this algorithm is to find pairs of incompatible classes of minterms for each row and then replace these incompatible classes of minterms with incompatible classes of columns(blocks of $P(B)$) which contain those minterms. These incompatible classes of columns are used to form the set of pairs of incompatible columns. Each pair of incompatible columns is represented by an edge in the incompatibility graph. We created two **GCA** algorithms, one for single output functions and one for multiple output functions; they share many steps. Multi-output GCA Algorithm has additional steps so that all columns are correctly classified for multiple output functions.

GCA Algorithm for Single Output Functions

The desired result of this algorithm is either an incompatibility graph or a compatibility graph. Fig. 3a shows the result of the first step in the algorithm. Here the cubes which are elements of each row are separated into classes based on the output value of the cubes. For single output functions, there are two output classes(0 and 1). For row 1, find two cubes which are elements of each output class(i.e., cubes 2 and 7 both have output value 1 and cubes 4 and 6 both have output value 0). Similarly, separate cubes in other rows into classes based on their output values. From these classes of cubes within each row, the observation can be made that columns which contain cubes in one class are incompatible with the columns that contain cubes in the opposite class. Therefore, with this

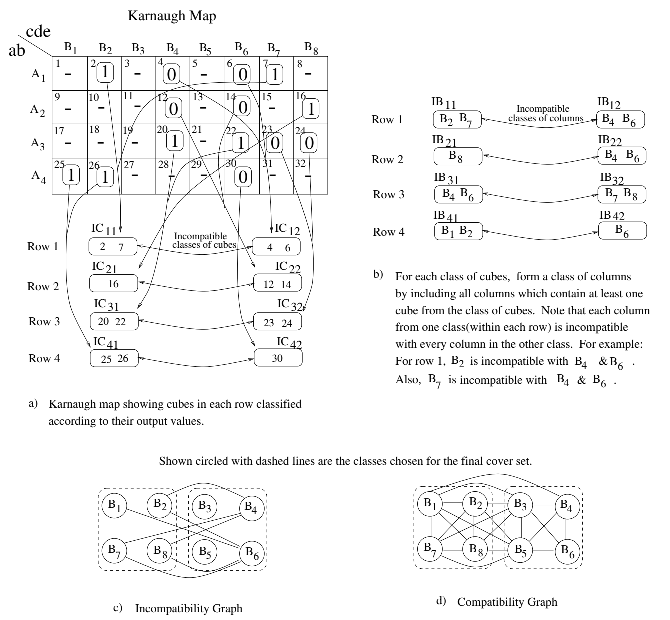


Figure 3: *Diagram illustrating* **GCA** *approach to Column Compatibility Checking for Single Output Functions*

observation in mind, perform the next step in the algorithm by forming a class of columns for each class of cubes. This is accomplished by including all columns which have at least one cube in common with a specific class of cubes to a new class of columns. The result of this step is shown in Fig. 3b. As shown in Fig. 3b, for classes within each row, all columns in one class are mutually incompatible with all columns in the opposite class. From these pairs of incompatible classes of columns, construct either an incompatibility graph or a compatibility graph.

For simplicity, first construct the incompatibility graph by adding an edge between nodes (columns) in the graph which are incompatible. This is done for each row as follows: for each column in one class, add an edge between the node that corresponds to that column, to every node corresponding to the columns in the other class. This results in the incompatibility graph shown in Fig. 3c. The complement of the incompatibility graph is shown in Fig. 3d.

Illustration Of GCA Approach On Function F2

Given is the function described by the Kmap in Fig. 3 and repeated again in Fig. 4, with the bound and free sets {c,d,e} and {a,b}, respectively. The following are the rough partitions for the bound set, free set, and output set, respectively. Commas separate minterm numbers(or cube numbers) within each rough partition and semicolons separate partition blocks. Don't cares are not enumerated in the partitions(i.e., they are not used in the partition operations) and $\emptyset$(empty set) indicates no

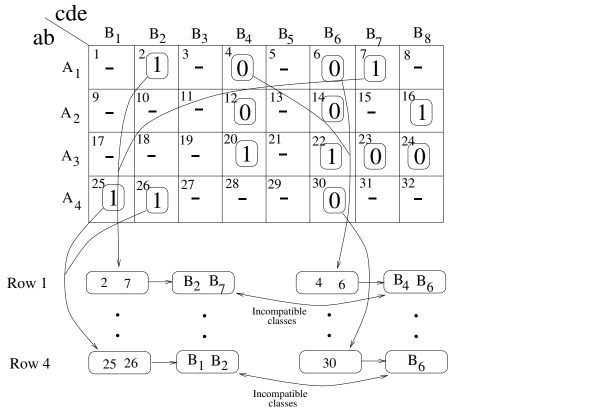


Figure 4: *KMap for function F2 showing incompatible classes of columns*

specified values in a particular partition. The goal is to construct the compatibility graph and perform column minimization to obtain a cover set Π_G [7].

$$\begin{aligned} P(B) &= (25;\ 2,26;\ \emptyset;\ 4,12,20;\ \emptyset;\ 6,14,22,30;\ 7,23;\ 16,24) \\ &= (B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8); \quad (1) \\ P(A) &= (2,4,6,7;\ 12,14,16;\ 20,22,23,24;\ 25,26,30) \\ &= (A_1, A_2, A_3, A_4); \quad (2) \\ P(F) &= (4,6,12,14,23,24,30;\ 2,7,16,20,22,25,26) \\ &= (F_1, F_2); \quad (3) \end{aligned}$$

Execution of the Single-Output GCA Algorithm.

Step (a). Generate set IC using function $FORM\_SET\_OF\_PAIRS\_IC(A, F, i = 4, j = 2)$. This function performs the following calculations:
$\forall i \forall j : IC_{ij} = A_i \cap F_j$.
The first class to be formed is IC_{11}.
It is formed as follows:
$IC_{11} = (A_1 \cap F_1) = ((2,4,6,7) \cap (4,6,12,14,23,24,30)) = (4,6)$.
Similarly, the remaining IC_{ij} are found. This results in the following classes of cubes; $IC_{11} = (4,6)$, $IC_{12} = (2,7)$, $IC_{21} = (12,14)$, $IC_{22} = (16)$, $IC_{31} = (23,24)$, $IC_{32}= (20,22)$, $IC_{41} = (30)$, $IC_{42} = (25,26)$.
The following is IC expressed as the set of pairs of incompatible classes of the form (IC_{i1}, IC_{i2}):
$IC = (((4,6),\ (2,7)),\ ((12,14),\ (16)),\ ((23,24),\ (20,22)),\ ((30),\ (25,26)))$.

Step (b). Generate IB using function $FORM\_SET\_OF\_PAIRS\_IB(IC, B, i = 4, n = 8)$. IB is found accordingly: $IB_{ij} = \{n \mid \forall i \forall j \ \ IC_{ij} \cap B_n \neq \emptyset\}$

For simplicity, only the column number is used to denote each column (i.e., B_i is shown as number i). The first class in IB to generate is IB_{11}. IB_{11} is found incrementally as follows(initially $IB_{11} = (\emptyset)$):

[i)] $IC_{11} \cap B_1 = (4,6) \cap (25) = \emptyset$,
therefore IB_{11} remains unchanged,
[ii)] $IC_{11} \cap B_2 = (4,6) \cap (2,26) = \emptyset$,
therefore IB_{11} remains unchanged,
[iii)] $IC_{11} \cap B_3 = (4,6) \cap (\emptyset) = \emptyset$,
therefore IB_{11} remains unchanged,
[iv)] $IC_{11} \cap B_4 = (4,6) \cap (4,12,20) = (4)$,
therefore $IB_{11} = (4)$,
[v)] $IC_{11} \cap B_5 = (4,6) \cap (\emptyset) = \emptyset$,
therefore IB_{11} remains unchanged,
[vi)] $IC_{11} \cap B_6 = (4,6) \cap (6,14,22,30) = (6)$,
therefore $IB_{11} = (4,6)$,
[vii)] $IC_{11} \cap B_7 = (4,6) \cap (7,23) = \emptyset$,
therefore IB_{11} remains unchanged,
[viii)] $IC_{11} \cap B_8 = (4,6) \cap (16,24) = \emptyset$,
therefore the resulting class for $IB_{11} = (4,6)$.
Similarly, the remaining IB_{ij} classes are found. This results in the following classes of columns: $IB_{11} = (4,6)$, $IB_{12} = (2,7)$, $IB_{21} = (4,6)$, $IB_{22} = (8)$, $IB_{31} = (7,8)$, $IB_{32} = (4,6)$, $IB_{41} = (6)$, $IB_{42} = (1,2)$.
Therefore
$IB = (((4,6),(2,7)), ((4,6),(8)), ((7,8),(4,6)), ((6),(1,2)))$.
Note: Each column in a pair of classes is incompatible with all columns in the opposite class. Each of these pairs represents the sets of columns which have a conflicting output in a particular row and therefore cannot be combined in the column minimization step.

Step (c). Construct the desired graph using function $FORM\_GRAPH\_FROM\_IB(IB, i = 4, B, n = 8)$. This function performs the following calculations:

For each row index i, assign all columns in class IB_{i1} as pairwise incompatible with all columns in class IB_{i2}.
For row 1: $(IB_{11} = (4,6)) \not\sim (IB_{12} = (2,7))$. Therefore columns 4 and 6 are incompatible with columns 2 and 7. This forms the incompatible pairs B_{24}, B_{26}, B_{47}, and B_{67}.
For row 2: $(IB_{21} = (4,6)) \not\sim (IB_{22} = (8))$. Therefore columns 4 and 6 are incompatible with column 8. This forms the incompatible pairs B_{48} and B_{68}.
For row 3: $(IB_{31} = (7,8)) \not\sim (IB_{32} = (4,6))$. Therefore columns 4 and 6 are incompatible with columns 7 and 8. This forms the incompatible pairs B_{47}, B_{48}, B_{67}, and B_{68}.
For row 4: $(IB_{41} = (6)) \not\sim (IB_{42} = (1,2))$. Therefore column 6 is incompatible with columns 1 and 2. This forms the incompatible pairs B_{16} and B_{26}.
The sets of incompatible pairs for rows results in the set of pairwise incompatible columns
$I_B = (B_{16}, B_{24}, B_{26}, B_{47}, B_{48}, B_{67}, B_{68})$.
The set IB of pairwise incompatible columns is used to form the incompatibility graph in Fig. 3c. The set of pairwise compatible columns C_B can be obtained simply by removing set I_B from the set of all pairs of columns:
$C_B = (B_{12}, B_{13}, B_{14}, B_{15}, B_{17}, B_{18}, B_{23}, B_{25}, B_{27}, B_{28}, B_{34}, B_{35}, B_{36}, B_{37}, B_{38}, B_{45}, B_{46}, B_{56}, B_{57}, B_{58}, B_{78})$

This set of pairwise compatible columns forms the compatibility graph shown previously in Fig. 3d. Covering of graph creates the cover set Π_G [7].

Application To Multiple Output Functions

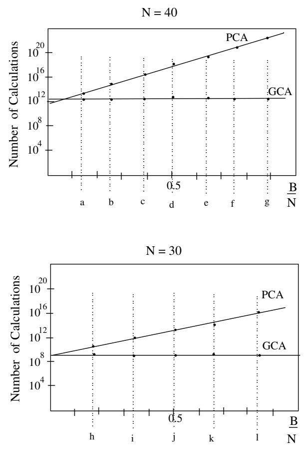
The fundamental distinction between **GCA** algorithms for single output vs. multiple output functions is that cubes may be compatible with more than one output class(output partition block) in multiple output functions. These cubes are referred to as repeated cubes. In single output functions, disjoint subsets of cubes are classified in each row according to output classes they belong to. When this is done, then pairs of classes of cubes are formed($IC_{ij}; IC_{ik}$). In each of these pairs of classes, all cubes in one class are incompatible with all cubes in the other class. However, in multiple output functions, cubes classified in the same manner may result in pairs of classes of cubes which are not disjoint. When this occurs, then all cubes in one class are not incompatible with all cubes in the other class. To solve this problem, the set of pairs of classes of incompatible cubes, denoted as IR, is formed first for the set of repeated cubes only. Then the repeated cubes are removed from all classes within set IC to form the new set IC of non-repeated cubes. The set IR is appended to IC and the remainder of the algorithm is identical to the single output algorithm [3].

5 Analysis Of The New Approach Versus The Classical Approach

In this section an analysis of the new approach(**GCA**) introduced in Sect. 4 versus the classical approach(**PCA**) introduced in Sect. 3 is presented. [1] To determine the number of calculations (intersection and union operations) required by each approach the following formulas were used:
For the **PCA** approach of Luba et al[8], the expression for finding pair-wise column compatibility is: $P(A) \cdot (B_i \cup B_j) \subseteq P(F)$ or more specifically, $A_k \cap (B_i \cup B_j) \subseteq F_m$. The number of required calculations is: **PCA** $= R \times O \times \binom{C}{2}$, where $\binom{n}{r} = \frac{n!}{r!\,(n-r)!}$. For the new approach(**GCA**), the expression for finding pairs of incompatible classes of columns is: $P(A) \cdot P(F) \cdot P(B)$ or more specifically, $A_k \cap F_m \cap B_i$. The number of calculations required is: **GCA** $= R \times O \times C$.
Variables are defined as follows: **PCA** = "Pair Compatibility Approach", **GCA** = "Group Compatibility Approach", A_k = individual blocks of the free set $P(A)$, B_n

[1] The following analysis is done only for single output functions. The reasons for this are:(1) because any multiple output function are replaced by multiple single output functions, (2) because the formula for single output functions for the new approach is much simpler to express mathematically.

	N	|A|	|B|	PCA	GCA
a	40	35	5	3.4×10^{13}	2.2×10^{12}
b	40	30	10	1.1×10^{15}	2.2×10^{12}
c	40	25	15	3.6×10^{16}	2.2×10^{12}
d	40	20	20	1.1×10^{18}	2.2×10^{12}
e	40	15	25	3.7×10^{19}	2.2×10^{12}
f	40	10	30	1.2×10^{21}	2.2×10^{12}
g	40	5	35	3.8×10^{22}	2.2×10^{12}



	N	|A|	|B|	PCA	GCA
h	30	25	5	3.3×10^{10}	2.1×10^{9}
i	30	20	10	1.1×10^{12}	2.1×10^{9}
j	30	15	15	3.5×10^{13}	2.1×10^{9}
k	30	10	20	1.1×10^{15}	2.1×10^{9}
l	30	5	25	3.6×10^{16}	2.1×10^{9}

Figure 5: *Plots of the two approaches represented by the formulas* **PCA** *and* **GCA** *for a constant total number of variables(N) and varying numbers of variables in the free set(A) and bound set(B).*

	N	|A|	|B|	PCA	GCA
m	15	10	5	1.0×10^{6}	6.5×10^{4}
n	20	10	10	1.1×10^{9}	2.1×10^{6}
o	25	10	15	1.1×10^{12}	6.7×10^{7}
p	30	10	20	1.1×10^{15}	2.1×10^{9}
q	35	10	25	1.1×10^{18}	6.9×10^{10}
r	40	10	30	1.2×10^{21}	2.2×10^{12}
s	45	10	35	1.2×10^{24}	7.0×10^{13}

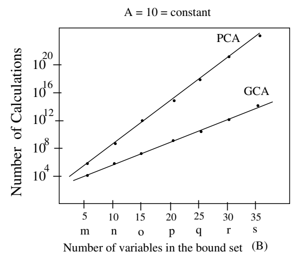


Figure 6: *Plots of the two approaches represented by the formulas* **PCA** *and* **GCA** *when the number of variables in the bound set are much greater than the number of variables in the free set*

= individual blocks of the bound set $P(B)$, F_m = individual blocks of the output set P_F, $\mid A \mid$ = number of variables in the free set, $\mid B \mid$ = number of variables in the bound set, $C = 2^{|B|}$ = number of columns in the bound set, $R = 2^{|A|}$ = number of rows in the free set, Y = number of output variables, $O = 2^{|Y|}$ = number of blocks in the output partition.

Fig. 5 presents plots of the two approaches represented by the formulas **PCA** and **GCA** for a constant total number of variables (N) with varying numbers of variables in the bound and free sets. Note that when the number of variables in the bound set is much larger than the number of variables in the free set, there exists a several orders of magnitude difference in the number of calculations (intersections and unions) required. Similarly, one can observe several orders of magnitude difference in the number of calculations when the number of variables in the bound set are much greater than the number of variables in the free set (Fig. 6).

Table 1: *User time spent calculating column compatibility on MCNC benchmarks using different approaches(***PCA** *vs* **GCA***) with varying sizes of bound sets, smaller than 12, usually 2, 3 or 4.*

Initial Function				GUD GCA	GUD PCA
Benchmark	inp	out	cub	t(s)	t(s)
5xp1	7	10	143	15	72
Z5xp1	7	10	141	19	75
adr2	4	3	24	0	0
b12	15	9	72	51	321
bw	5	28	97	6	17
c8	28	18	166	9	10
cc	21	20	96	18	3
con1	7	2	18	2	3
ex5	8	63	214	210	2303
f51m	8	8	154	38	165
misex1	8	7	40	10	18
rd53	5	3	63	1	2
rd73	7	3	274	20	120
rd84	8	4	515	229	787
root	8	5	256	121	803
squar5	5	8	56	1	2
xor5	5	1	32	0	0

Table 2: *Summary of Results for Table 1*

	Category					
Program	A(s)	B(s)	C(s)	D	E	F
GUD(GCA)	750	44	229	15	88%	1
GUD(PCA)	4,701	277	2,303	1	6%	0

Categories (in all Summmary of Results tables): A-Total Time(all benchmarks). B-Average Time per benchmark. C-Maximum Time for any benchmark. D-Number of times an algorithm had the lowest user time(including ties). E-F-Number of times an algorithm had a user time which was at least one order of magnitude(x10) faster than the competing algorithm. inp - number of inputs, out - number of outputs, cub - number of cubes, t(s) - time in seconds.

The analysis of the two approaches illustrates dramatic differences in the number of calculations required by each of the approaches when the number of variables in the bound set is large. This suggests a potential for significant savings using the new approach.

6 Experimental Results

This section compares the execution times for both algorithms. In Sect. 6.1, results of complete decompositions on the MCNC benchmarks are compared for **PCA** and **GCA** algorithms. In Sect. 6.2, comparison results are shown on the partial decompositions of FLASH benchmarks from Wright Labs with fixed bound set sizes. Versions of program GUD used for decomposition in the comparisons are:

GUD(**GCA**): Version of GUD using the new GCA algorithm to calculate the column compatibility.

GUD(**PCA**): Version of GUD using a commonly used PCA algorithm to calculate the column compatibility. PCA is used in the program DEMAIN [7].

Table 3: *Time spent calculating column compatibility on FLASH benchmarks using two different methods(***PCA** *vs* **GCA***) with two variables in the bound set.*

Initial Function				GUD PCA	GUD GCA
Benchmark	inp	out	cub	t(s)	t(s)
psu_add0_90	12	1	410	0	0
psu_add2_90	12	1	410	1	1
psu_add4_90	12	1	410	2	0
psu_contains_4_ones_90	12	1	410	1	0
psu_greater_than_90	12	1	410	1	1
psu_interval1_90	12	1	410	1	0
psu_interval2_90	12	1	410	1	0
psu_majority_gate_90	12	1	410	1	1
psu_pal_90	12	1	410	1	0
psu_parity_90	12	1	410	1	1
psu_sim12_90	12	1	410	1	0
psu_substr1_90	12	1	410	1	0
psu_subtraction1_90	12	1	410	0	1
psu_subtraction3_90	12	1	410	1	0

Table 4: *Summary of Results for Table 3*

	Category						
Program	A(sec)	B(sec)	C(sec)	D	E	F	G
GUD(**GCA**)	5	0.3	1	13	93%	0	0
GUD(**PCA**)	13	0.9	2	6	43%	0	0

6.1 Comparison Between the PCA and GCA Algorithms in the General Decompositions of the MCNC Benchmarks

From the results in Tables 1 and 2 it can be observed that the **GCA** approach clearly outperforms the **PCA** approach in terms of execution time, on nearly every benchmark example. The execution time for each approach is the total time spent on calculation of the column compatibility throughout the complete decomposition process on each benchmark. A benchmark may require several subfunctions to be computed in the decomposition process and each subfunction may require many graphs to be constructed by either approach before a bound set is selected which yields an acceptable decomposition. It is important to note that each approach constructs identical graphs on each benchmark. The only difference in the whole decomposition process is which approach is used to construct the graphs. The partitioning strategy always tries small bound sets first and if no decomposition is found then the number of variables in the bound set is increased by one. The bound set size was limited to twelve. However, the size of the most bound sets which resulted in an acceptable decomposition were either two or three variables. This is a significant point to make in the comparison of the different approaches because the new approach(**GCA**) was expected to have a significant advantage when bound sets are large. The results show, however, that the new approach is substantially faster even for small bound sets.

For a number of examples, the highest quality decomposition is for large bound sets. Therefore, the **GCA** requires less computation execution time even on small bound sets and provides the capability required to check large bound sets which can't be feasibly checked by the **PCA** approach.

Table 5: *Time spent calculating column compatibility on FLASH benchmarks using two different methods(***PCA** *vs* **GCA***) with five variables in the bound set.*

Initial Function				GUD PCA	GUD GCA
Benchmarks	inp	out	cub	t(s)	t(s)
psu_add0_90	12	1	410	31	3
psu_add2_90	12	1	410	29	2
psu_add4_90	12	1	410	28	3
psu_contains_4_ones_90	12	1	410	31	1
psu_greater_than_90	12	1	410	33	2
psu_interval1_90	12	1	410	37	0
psu_interval2_90	12	1	410	35	1
psu_majority_gate_90	12	1	410	33	2
psu_pal_90	12	1	410	38	0
psu_parity_90	12	1	410	31	2
psu_sim12_90	12	1	410	32	1
psu_substr1_90	12	1	410	46	1
psu_subtraction1_90	12	1	410	35	2
psu_subtraction3_90	12	1	410	36	2

Table 6: *Summary of Results for Table 5*

	Category						
Program	A(sec)	B(sec)	C(sec)	D	E	F	G
GUD(**GCA**)	22	1.6	3	14	100%	14	0
GUD(**PCA**)	475	33.9	46	0	0%	0	0

6.2 Comparison Between PCA and GCA Algorithms on FLASH Benchmarks with Fixed Bound Set Sizes

This section compares the results of the **PCA** and **GCA** algorithms when the number of variables in the bound set is fixed. For example, if the number of variables in the bound set is specified to be 10, then only the bound sets of size 10 are used. In Tables 3-7, only two graphs were constructed for each benchmark. The purpose was to control the decompositions so that comparisons could be made, not only between the each algorithm used, but also between execution times for various bound set sizes.

Observe that there is very little difference in execution times when the number of variables in the bound set is 2 (Table 3). However, there is a substantial difference in execution times (Table 5). The **GCA** approach consistently outperforms the **PCA** approach when the number of variables in the bound set is 5. In fact, the **GCA** approach outperforms the **PCA** approach by more than an order of magnitude in execution time on every benchmark. In Table 7, an even more dramatic difference can be observed between the execution times of the two approaches. When the number of variables in the bound set is equal to 10, the **GCA** approach outperforms the **PCA** approach by more than two orders of magnitude on every benchmark! A Summary information on the comparisons made can be found in the corresponding Tables 4, 6, and 8.

Table 7: *Time spent calculating column compatibility on FLASH benchmarks using two different methods(***PCA** *vs* **GCA***) with ten variables in the bound set.*

Initial Function				GUD(PCA)	GUD(GCA)
Benchmark	inp	out	cub	t(s)	t(s)
psu_add0_90	12	1	410	358	1
psu_add2_90	12	1	410	202	1
psu_add4_90	12	1	410	141	1
psu_contains_4_ones_90	12	1	410	147	1
psu_greater_than_90	12	1	410	148	1
psu_interval1_90	12	1	410	142	0
psu_interval2_90	12	1	410	163	1
psu_majority_gate_90	12	1	410	115	1
psu_pal_90	12	1	410	114	1
psu_parity_90	12	1	410	130	2
psu_sim12_90	12	1	410	115	1
psu_substr1_90	12	1	410	122	1
psu_subtraction1_90	12	1	410	119	1
psu_subtraction3_90	12	1	410	127	1

Table 8: *Summary of Results for Table 7*

	Category						
Program	A(sec)	B(sec)	C(sec)	D	E	F	G
GUD(**GCA**)	14	1	2	14	100%	14	14
GUD(**PCA**)	2143	153	358	0	0%	0	0

7 Conclusions

The results from Tables 3-8 clearly show that the **GCA** approach strongly outperforms the **PCA** approach in execution time without sacrifying the result's quality. The **GCA** algorithm performs much better than the **PCA** algorithm when larger bound sets are used in the decomposition process. When the bound set is large enough(five variables or more), the **GCA** approach outperforms the **PCA** approach by orders of magnitude in execution time. We expected that the **GCA** algorithm will be faster than the **PCA** algorithm when larger bound sets were used, but did not expect such dramatic differences and such high consistency with the pre-testing analysis (Figs. 6 and 7).

Much more efficient column compatibility checking results in much faster decomposition process. However, it is even more important that the **GCA** algorithm can be used to create the compatibility graph for larger bound sets with little or no increase in the execution time. By being able to use larger bound sets, the search space of feasible decompositions is increased, thereby making it possible to find better decompositions. Large bound sets create on average more overlaps when **multi-coloring** is used instead of coloring of the incompatibility graphs. Thus, as demonstrated in [9], they lead more often to decomposition of both blocks G and H of the decomposition being relations, when the relational decomposition is applied [10, 4]. In addition to saving time when the bound sets are large, this new approach can be used to search a significant part of the search space on large functions not feasible using previous approaches due to the large computational requirements. The new approach can be integrated into a modified graph coloring algorithm to speed up column minimization as well. Our future research involves experimental comparison of decomposition quality with large and small bound sets.

References

[1] R. L. Ashenhurst. "The Decomposition of Switching Functions." *Proc. Intern. Conf. Comp. Aid. Des.*, pp. 84-87, Nov. 1959.

[2] H.A. Curtis. "A New Approach to the Design of Switching Circuits," D. Van Nostrand Company, 1962.

[3] M. Burns, M.S. Thesis, PSU, 1997.

[4] 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*, pp. 287-292.

[5] L. Jozwiak, "General Decomposition and Its Use in Digital Circuit Synthesis," VLSI Design: An Intern. J. of Custom Chip Design, Simulation and Testing, Vol. 3., No. 3-4, 1995.

[6] F.A.M. Volf, L. Jozwiak, and M.P.J. Stevens, "Division-Based versus General Decomposition-Based Multiple-Level Logic Synthesis," VLSI Design: An Intern. J. of Custom Chip Design, Simulation and Testing, Vol. 3., No. 3-4, 1995.

[7] T. Luba, M. Mochocki, J. Rybnik, "Decomposition of Information Systems Using Decision Tables," *Bull. Pol. Ac. Sci.* Vol. 41, No.3, 1993.

[8] T. Luba, R. Lasocki, "Decomposition of Multiple-valued Boolean Functions," *Applied Mathematics and Computer Science,* Vol.4, No.1, pp. 125-138, 1994.

[9] R. Malvi, "Efficient Algorithms for Column Compatibility Problem in Boolean Decomposition", M.S. Thesis, PSU, 1997.

[10] 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*, pp. 13-18.

[11] M. Perkowski, M. Chrzanowska-Jeske, and Y. Xu, "Multi-Level Programmable Arrays for Sub-Micron Technology Based on Symmetries," *Proc. ICCIMA'98*, pp. 707-720, Febr. 1998, World Scientific, Signapure.

[12] W. Wan, 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,* Sept. 7-10, Hamburg, 1992, pp. 230 - 235.