FAST APPROXIMATE MINIMIZATION OF SUM-OF-CDEC MULTI-OUTPUT STRONGLY UNSPECIFIED BOOLEAN FUNCTIONS

Sanof Mohamed, Marek Perkowski +, and Lech Jozwiak †,

COMPASS Design Automation, San Jose, USA

+ Dept. of Electrical Engn.,

Portland State University, Portland, OR, USA

† Faculty of Electrical Engineering,

Eindhoven University of Technology, The Netherlands.

TALK OUTLINE

1. CY7C361 EPLD of Cypress Semiconductor
2. CDEC Gate
3. CDEC-PLA, Sum-of-CDEC Architecture and Optimization Problem
4. CDEC Implicants
5. Conditionally compatible cubes
6. Compatible Graph Coloring Problem for finding best CDEC covering of single output functions
7. Conditional Coloring algorithm to solve this problem
8. Splitting large cubes as preprocessing to Conditional Coloring.
9. Generalization to Multi-output functions.
10. Experimental results.
11. Conclusions

CY7C361 CHIP FROM CYPRESS

- **A Programmable Logic Device (PLD) from Cypress Semiconductor**, CY7C361, has been optimized for the realization of very fast asynchronous (self-synchronized) sequential controllers by means of concurrency and partitioning of Finite State Machines (FSMs).
- This approach leads to Tokenized State Machines (TSM), a new concept which generalizes several known notions of micro-parallelism and was used to synthesize a number of practical controllers (Coppola 92).
- Controller can be designed from multiple deterministic and non-deterministic finite state machines, and mapped into this new device.

SUM OF CDEC ARCHITECTURE IN CY7C361 CHIP

- The input array of CY7C361 consists of an array of logic gates feeding, in a one-for-one fashion, the state macrocell array.
- Each logic gate in the array, called a **Condition Decoder** (CDEC), is the product of a product term, and the complement of a product term.

  CDEC = $P \cdot \overline{Q}$, where P and Q are products of literals
- The inputs to each gate come from any external input to the device and from a number of feedback connections from macrocells, arranged in a segmented array.
- The excitation functions of the state macrocells are **Sum of CDEC** (SOC) expressions.

DEVICE FITTING PROBLEM IN EPLD CHIPS

- The CDEC gate has been shown to be an excellent choice for a very fast realization of conditions to state macrocells. (simple branch and join conditions of concurrent FSMs).
- A problem that is related to the **"device fitting problem"** of the Cypress CY7C361 chip is the **SOC Minimization**.
- Since the CY7C361 has only 32 macrocells, a **high quality logic minimization** to reduce the number of macrocells is very important.
- Real-life examples demonstrated that the SOC minimization can often transform a non-fittable netlist into a fittable one.

IMPROVED APPROACH FOR SOC MINIMIZATION

- The SOC minimization problem was solved as a part of **WARP**, a Cypress Corporation's PLD synthesis system based on behavioral VHDL.
- Here we present a CDEC minimizer **superior** to that in WARP.
- The goal of this paper is, however, more general. We believe that the CDEC gate minimization can find applications both in **custom standard cell-based design**, and in **programmable structures**.
- CDEC gate can be used as a **general-purpose complex gate** in layout generators for custom design layout.

CDEC PLA

- New PLD structures, such as the **CDEC-PLA** proposed below.
- The Cypress chip is therefore not the sole motivation of our work, and this paper presents the SOC minimization as a **generic logic synthesis problem**.
- Our algorithms are especially good for **very strongly unspecified functions** defined by ON and OFF sets.
- Applicable in **Machine Learning** (ML) and **Knowledge Discovery from Data Bases** (KDD).
- **CDEC-PLA** realizes multi-output SUM-OF-CDEC (SOC) expressions, which generalize the well known SUM-OF-PRODUCTS expressions of PLAs.

THE NEW PROBLEM OF CDEC-PLA MINIMIZATION

1. We give efficient heuristic algorithm for the minimization of the **multi-output** SOC expressions.
2. Comparison of several versions of the algorithm on benchmarks proves that our fast multi-output algorithm generates **high-quality solutions**.
3. This algorithm produced the minimum solutions on all small single-output functions, as required by this chip.
4. It is especially efficient on strongly unspecified functions.
5. A new approach for solving covering/factorization problems, called **Conditional Graph Coloring** will be proposed.
6. It can be used in other minimization algorithms for circuits with few logic levels.

THREE-LEVEL MINIMIZATION

- Tsutomu Sasao proved that for "nearly all functions" three levels is enough.
- This result should stimulate research on various three-level minimizers.
- This paper introduces three-level OR/AND/OR circuits implemented as Sum of CDECs.

CONDITIONAL GRAPH COLORING APPROACH

- An original approach to solve SOC covering problems **without** generating prime CDEC-implicants.
- The method is based on a new type of graph coloring algorithm.
- Instead of solving the covering problem with prime CDEC-implicants, we solve the coloring problem for a graph whose nodes correspond to minterms or some product implicants of a new type.
- Graph-coloring problems can be solved using:
  1. **heuristic**, non-backtracking algorithms for an approximate minimization,
  2. **optimal** backtracking algorithms for an exact solution with the minimum number of implicants.

CDEC IMPLICANTS

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

A CDEC-implicant can be realized with a single CDEC gate.

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

- Product terms and prime implicants of SOP are special cases of CDEC-implicants.
- A prime SOP implicant is not necessarily a prime CDEC, because it can be included in a CDEC-implicant.

SOC MINIMIZATION PROBLEM AS CDEC COVERING

- A standard PLA is realized from two planes: an input AND plane and an output OR plane.
- **CDEC-PLA** is composed of a standard output *OR Plane* and a *CDEC Plane* that replaces the AND plane of standard PLAs.
- The CDEC plane is composed of CDEC gates, the outputs of which are the inputs to the OR plane.
- **SOC minimization** is based on covering minterms with **CDEC-implicants**, called **CDEC-covering**.
- CDEC gate $P \cdot \overline{Q}$ has an **AND-part** P and a **NAND-part** $\overline{Q}$.
- The concept of a CDEC-implicant is a generalization of a prime implicant (a prime) and a product implicant.

SOC MINIMIZATION WITH PRIME CDEC-IMPLICANTS

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

- CDEC is realized by a single gate.
- The **SOC Minimization** process results in a **SOC expression.**
- The exact SOC expression of function f has never more gates than a SOP of this function because the set of all CDEC implicants includes the set of all prime implicants.

THE SOC MINIMIZATION APPROACH FROM WARP

(1) Generate **all prime implicants** of function f (using Espresso).

(2) **Factorize** the prime implicants to prime CDEC-implicants.

(3) Solve the **covering problem** with true minterms as columns and prime CDEC-implicants as rows, such that the number of CDEC-implicants is minimal.

SOC MINIMIZATION APPROACH FROM WARP HAS DRAWBACKS:

(1) Since not all prime CDEC-implicants can be generated by factoring prime implicants, the **minimum** CDEC-covering with prime CDEC-implicants will be **lost**.

(2) The generation of prime CDEC-implicants based on standard factoring procedures is not efficient.

(3) Since the number of prime CDEC-implicants is much larger than that of prime implicants, a large covering table is created, which **cannot be solved** even for those functions that the exact minimal prime covering would be still possible to find.

(4) Using **minterms** in the table **limits** the size of problems that can be solved.

EXAMPLE FOR THE CRITIQUE OF THE APPROACH FROM WARP.

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

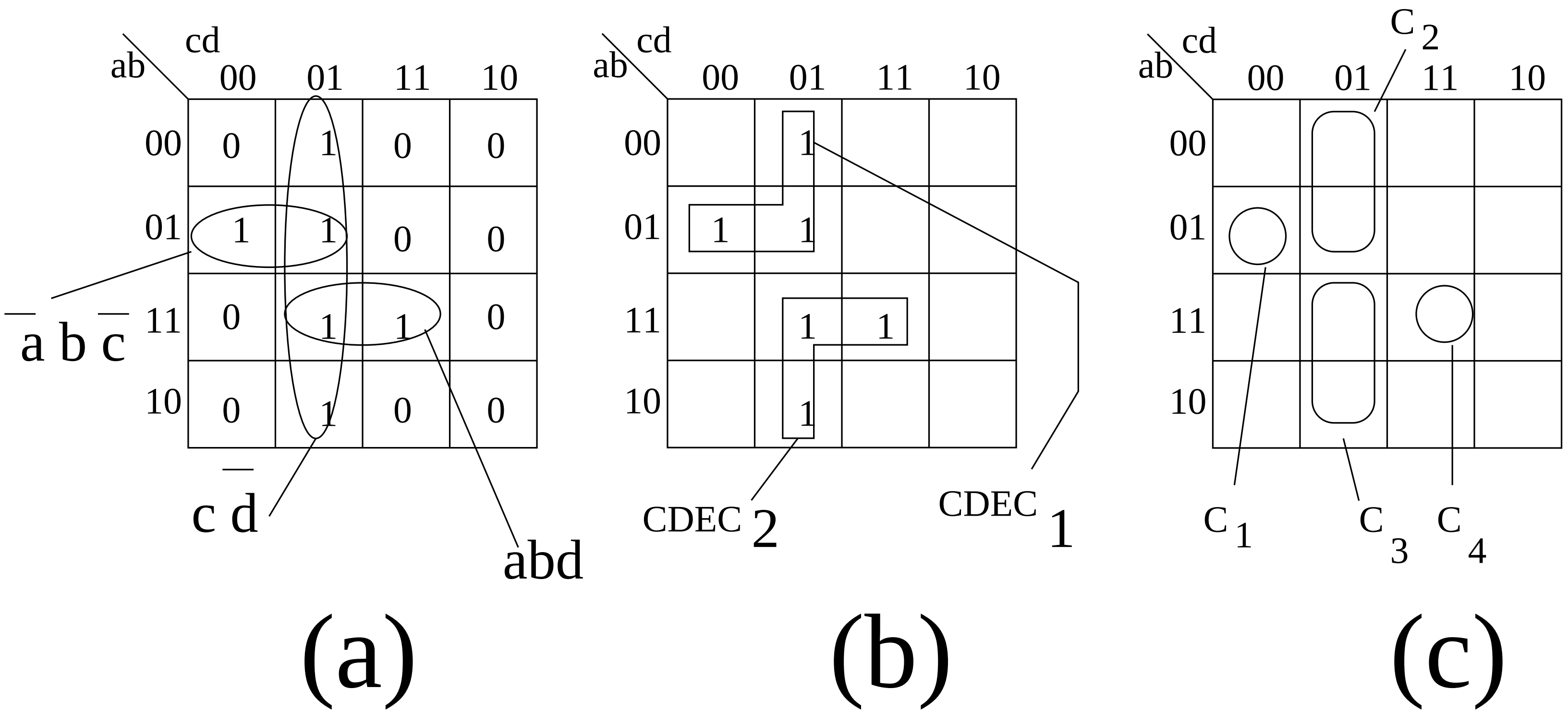


Figure 1: *(a,b) Function that demonstrates that the minimum SOC cannot be obtained by factorization of primes, (c) Compatible and conditionally compatible cubes. Cubes C_2 and C_3 are compatible, cubes C_1 and C_2 are conditionally compatible, cubes C_1 and C_4 are incompatible.*

- A function for which the approach based on factoring prime implicants can not find the minimum CDEC cover was shown.
- The function f can be represented as the sum of **three** essential prime implicants: $f = \bar{c}d + \bar{a}b\bar{c} + abd$**.**
- Based on these implicants, *f* can not be further factorized to CDEC-implicants.
- The CDEC realization would consist of **3 CDEC gates** and the CDEC-implicants would be identical to the prime implicants.
- However, function f can be realized with **only two CDEC gates**. The minimal solution can be obtained by the **factorization of product implicants that are not primes**, as shown in Figure.

  $f = \bar{a}b\bar{c} + \bar{a}\bar{c}d + a\bar{c}d + abd = \bar{a}\bar{c}(b + d) + ad(\bar{c} + b) = \bar{a}\bar{c}\overline{(\bar{b}\bar{d})} + ad\overline{(c\bar{b})} = CDEC_1 + CDEC_2.$

TWO IMPORTANT PRINCIPLES CAN BE LEARNED FROM THIS EXAMPLE:

(1) Prime CDEC-implicants must be created not only from prime implicants, but also from some product implicants **included** in prime implicants.

(2) Essential prime implicants are useful to create essential prime CDEC-implicants, that are included in the **exact minimum CDEC cover**.

HOW TO USE THESE PRINCIPLES

- Observation (1) can be used in an algorithm to create all prime CDEC-implicants.
- Observation (2) can be used in both exact and approximate algorithms to find CDEC coverings more efficiently by first finding the essential and secondary essential CDEC-implicants, and thus reducing the combinatorial search.
- Our approach is based on graph coloring, rather than on covering, but it still generates implicants **starting from the essential impliants**, and next the secondary essential implicants.
- The advantage of coloring is that the **large sets of CDEC implicants are never generated** to be stored at once in memory.

POSSIBLE EXACT MINIMIZER BASED ON STANDARD APPROACHES:

(1) Find all prime CDEC-implicants.

(2) Find the covering of all true minterms with prime CDEC-implicants, that minimizes the number of CDEC-implicants in the cover (or the total cost of literals in these CDEC-implicants).

- This kind of exact approach could be used only for functions with a small number of input variables.
- The number of prime CDEC-implicants increases rapidly with the number of minterms, especially for functions with many don't cares.

A CRITIQUE OF SUCH AN APPROACH

- 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 (Brayton).
- A similar property can be shown for prime CDEC-implicants.
- Moreover, the number of CDEC-implicants is even more dramatically large.
- Thus, the application of algorithms based on generating **all** primes or **all** prime CDEC-implicants is **limited** because of results of this kind.

OUR APPROACH TO THE COMPATIBLE COLORING PROBLEM

- The graph for coloring, called **Graph for CDEC-Compatible Coloring** (GCCC), can be created with arbitrary cubes from the ON-set of the function f as nodes.
- For instance: minterms, primes, disjoint product implicants. minimal product implicants, or the (optimal) split cubes.
- 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 CONCEPT OF CUBE SPLITTING

- The advantage of using arbitrary cubes instead of minterms is the improved execution speed, but the minimum CDEC cover can be lost.
- The **cube splitting** algorithm creates the graph from **any** type of cubes.
- We use special **split cubes**, for their efficiency and solution quality.
- For split cubes, the minimal solution can be in theory lost, but experimental results show that this never happened on the tested benchmarks.

SUPERCUBE OPERATION FOR CONDITIONALLY COMPATIBLE CUBES

- Boolean functions in our system are represented as ON and OFF arrays (sets) of cubes.
- Set DC is implicit in our approach and only ON and OFF are stored.
- The supercube of two cubes C_i and C_j is denoted by $C_i \uplus C_j$.
- When the positional cube notation is used, the supercube operator corresponds to the component-wise Boolean OR of the two cubes.
- This is the smallest cube that includes both C_i and C_j.
- **Example:** $C_i = X101$, $C_j = X110$, $C_i \uplus C_j = X1XX$ is the supercube of C_i and C_j.
- We say that **two cubes overlap** if their intersection cube is non-empty.

COMPATIBLE CUBES

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

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

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

COMPATIBLE CUBES. II

Definition. 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 $SOFFP(SU_{ij})$.

CONDITIONALLY COMPATIBLE CUBES

Definition. Two cubes C_i and C_j are called **conditionally compatible** if cube $SOFFP(SU_{ij})$ does not intersect C_i nor C_j.

Such cubes can be combined to one CDEC-implicant $(SU_{ij} \cdot \overline{SOFFP(SU_{ij})})$, with cube SU_{ij} as the AND-part and expression $\overline{SOFFP(SU_{ij})}$ as the NAND-part.

The cube $SOFFP(SU_{ij})$ is called the **condition cube** under which C_i and C_j are conditionally compatible.

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

CONDITIONALLY COMPATIBLE CUBES. II

Definition The set Π of cubes is called a **set of compatible cubes** if each pair $C_i, C_j \in \Pi$, is either compatible or conditionally compatible with respect to the same condition cube $SOFFP(\biguplus_{C_i \in \Pi} C_i)$.

The set Π can be described by a single CDEC-implicant:

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

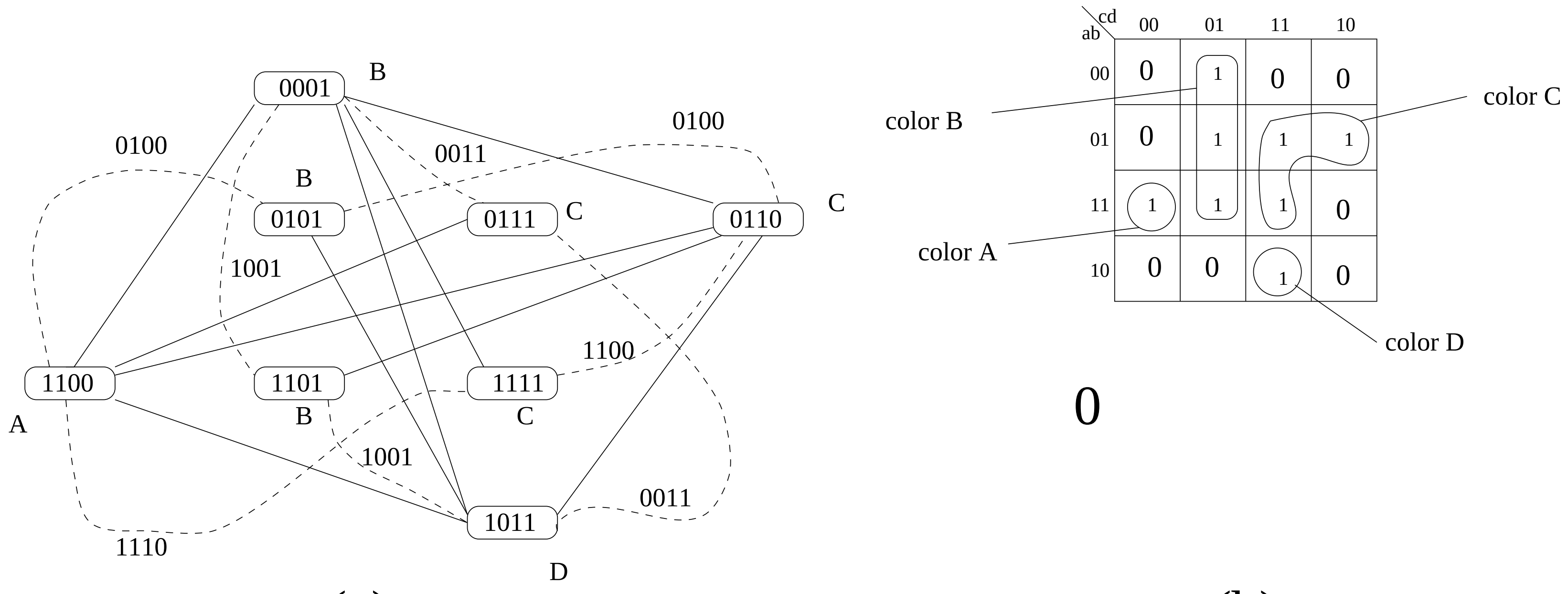
ALGORITHM FOR CONDITIONAL GRAPH COLORING

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

ALGORITHM FOR CONDITIONAL GRAPH COLORING. II

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

EXAMPLE FOR EXPLANATION OF THE COMPATIBLE COLORING CONCEPTS



(a) (b)

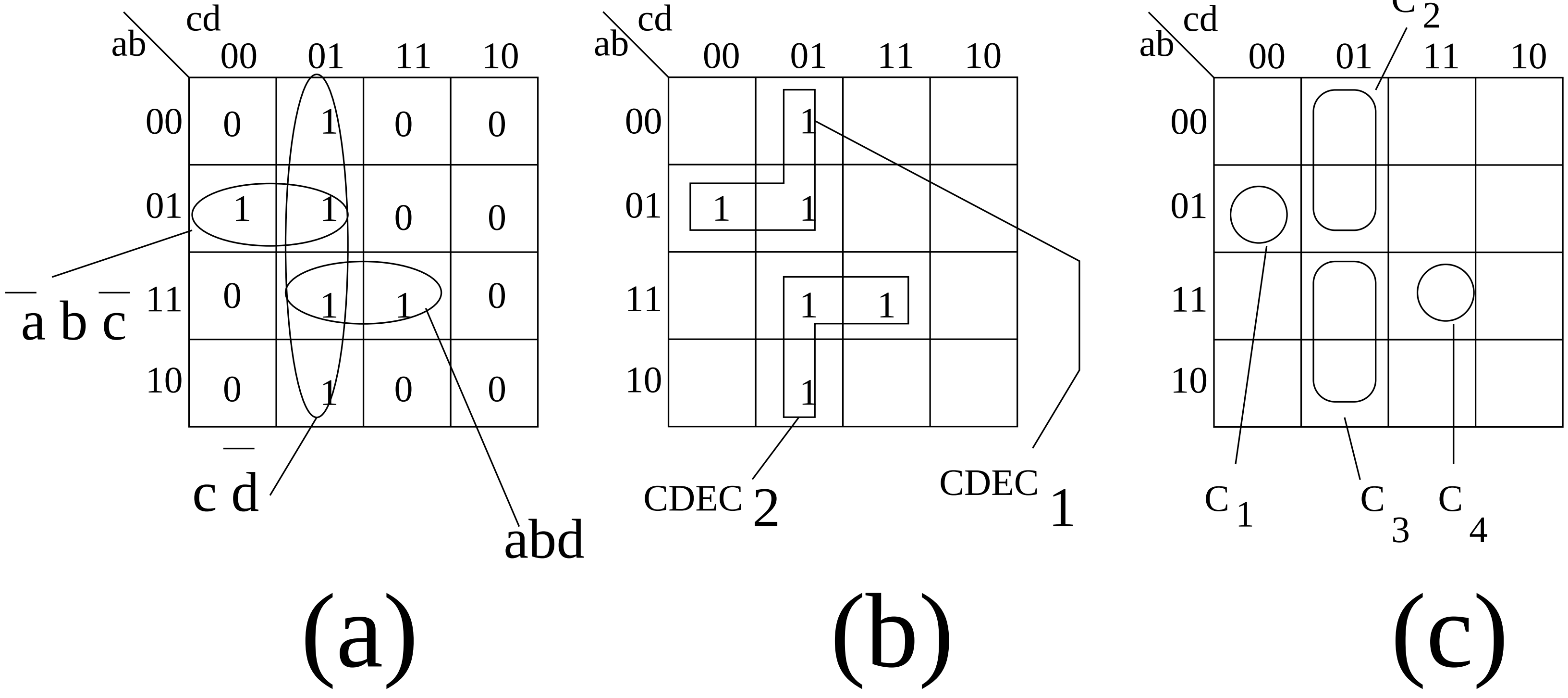
(a) graph GCCC for Examples 4 and 6 with a CDEC-minimum compatible coloring shown near nodes,

(b) minimum CDEC-implicant cover corresponding to the compatible coloring found in (a).

COMPATIBLE COLORING CONT.

- The dotted edges labeled with a cube are the conditional edges.
- 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.
- The solution found from the minimal conditional coloring, $g = ab\overline{c}\overline{d} + \ \overline{c}d\overline{a\ \overline{b}} + \ bc\overline{a\ \overline{d}} + \ a\overline{b}cd$, has the minimum number of terms, but not necessarily the minimum number of literals.
- Another coloring could find the minimum literal cost solution as $g' = \overline{a}d\overline{c} + \overline{a}bc + ab\overline{c} + bcd$.
- Thus, the literal cost should be calculated by the graph-coloring procedure as a secondary cost for all solutions which have the same number of colors.

ANOTHER EXAMPLE TO EXPLAIN COMPATIBLE COLORING



OBSERVATION

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

DEFINITIONS FOR COMPATIBLE COLORING

Definition A **compatible coloring** of each node with a single color is a coloring of nodes such that any two nodes linked by a normal edge have different colors and any set of nodes colored with the same color is a compatible set of nodes.

A **compatible set of nodes** is a set in which the nodes correspond to a set of compatible cubes.

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

DEFINITIONS FOR COMPATIBLE COLORING. II

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

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

COMPATIBLE MULTI-COLORING

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

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

EXACTNESS THEOREM

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

THEOREM 1. If a graph GCCC is created with minterms as nodes and the (multi)coloring of the GCCC is compatible and has the minimum number of colors (is an exact compatible (multi)coloring), then the CDEC covering created from this coloring has the **exact minimum number of CDEC-implicants**.

COMPATIBLE COLORING IN SOCMIN

In program SOCMIN, we first use Algorithm for optimal splitting of initial cubes to smaller cubes for which optimal compatible coloring could be found more efficiently.

The design objectives of this algorithm were:

- **(1)** size of problems that can be handled,
- **(2)** speed,
- **(3)** ability to find optimal solutions for non-cyclic functions.

Although it works with any functions, its advantages are clearly seen for strongly unspecified and non-cyclic functions.

Interestingly, a characteristic of our approach is that *"the more unspecified is the function, the better are our results, comparing to the known tools"*.

REALIZATION OF COMPATIBLE COLORING ALGORITHM.

- The basic idea of GCCC coloring algorithm is to color successively, the most restricted nodes with the most adjacent **normal edges**.
- By that, colors are chosen, that result in the smallest number of restrictions for all other nodes.
- It is known that non-cyclic functions are found quite frequently in real-life applications, Brayton.
- A minterm from an essential implicant that is covered by only this implicant has the most adjacent edges in the graph, so it will be colored first by our algorithm.

ADVANTAGE FOR NON-CYCLIC FUNCTIONS

- Although this is a graph-coloring algorithm, its operation is in essence similar to the well-known SOP-minimizing algorithm that extracts the essential and next (iteratively) the secondary essential primes, thus being able to find the exact solution (without setting the covering table) for non-cyclic functions.
- Therefore, our graph coloring algorithm gives very good results on non-cyclic functions.

REALIZATION OF COMPATIBLE COLORING ALGORITHM.

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

Definition. A node MI_i of the Graph GCCC is associated with the **Color-Restriction Vector** that includes all the restrictions of node MI_i. The color restrictions result from the coloring of other nodes of the Graph and are the colors that node MI_i is not allowed to be colored with.

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

SELECTING GOOD NODES AND COLORS

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

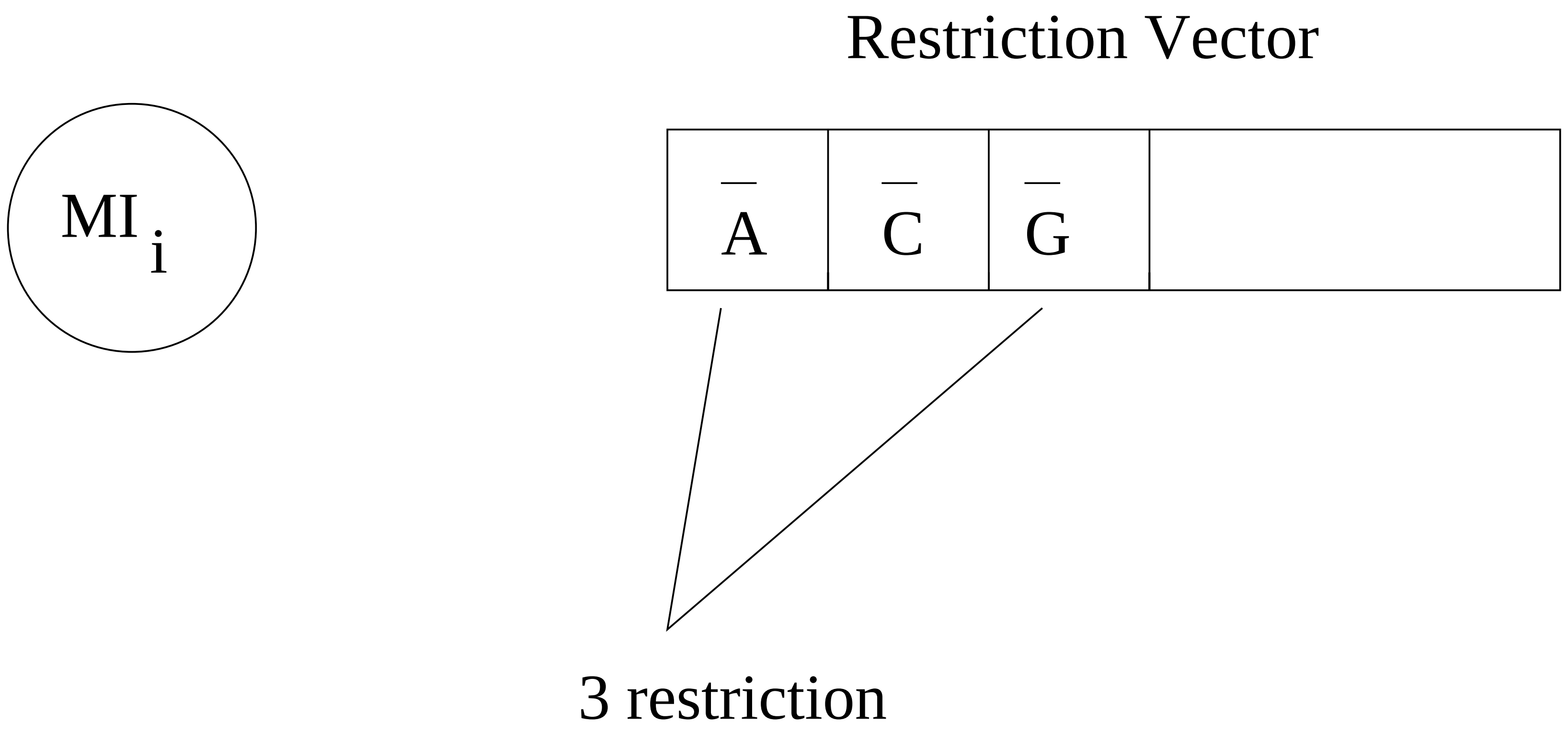
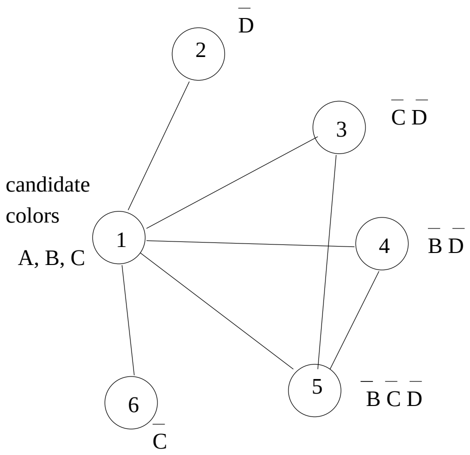


Figure 2: *Example of a node MI_i with a 3-color-Restriction. Node MI_i cannot be colored with colors A, C, or G.*

COLOR RESTRICTION MATRIX



(a)

C \ k	0	1	2	3	4
A	0	0	2	2	1
B	0	0	2	1	0
C	0	0	1	1	0

(b)

(a) candidates for coloring,

(b) resulting Color-Restriction Matrix for Candidate node MI_1.

EVALUATION OF CANDIDATE NODES FOR COLORING

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

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

(2) the degree of normal edges,

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

(4) the degree of conditional edges.

CANDIDATE COLORS.

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

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

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

REALIZATION OF COMPATIBLE COLORING ALGORITHM. COLOR RESTRICTIONS.

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

The selection of nodes to be colored next is done according to the minimal Max-Resulting-k-Restriction.

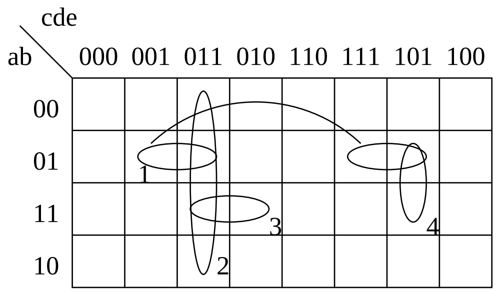
This means, that this is the first criterion to choose the next node and its color.

If there is an equality in the Max-Resulting-k-Restriction, then all the other k-Color-Restrictions of the Resulting Color Restriction Matrix are taken into account.

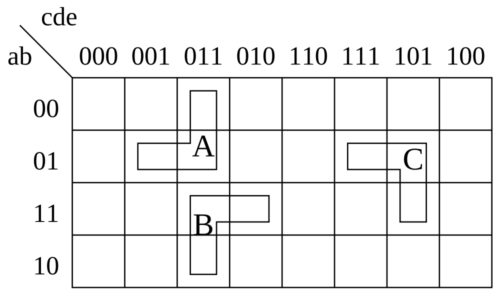
SPLITTING OF CUBES

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

SPLITTING OF PRIME CUBES IS NECESSARY TO OBTAIN MINIMUM CDEC COVER



(a)



(b)

IDEAS USED IN CUBE SPLITTING ALGORITHM

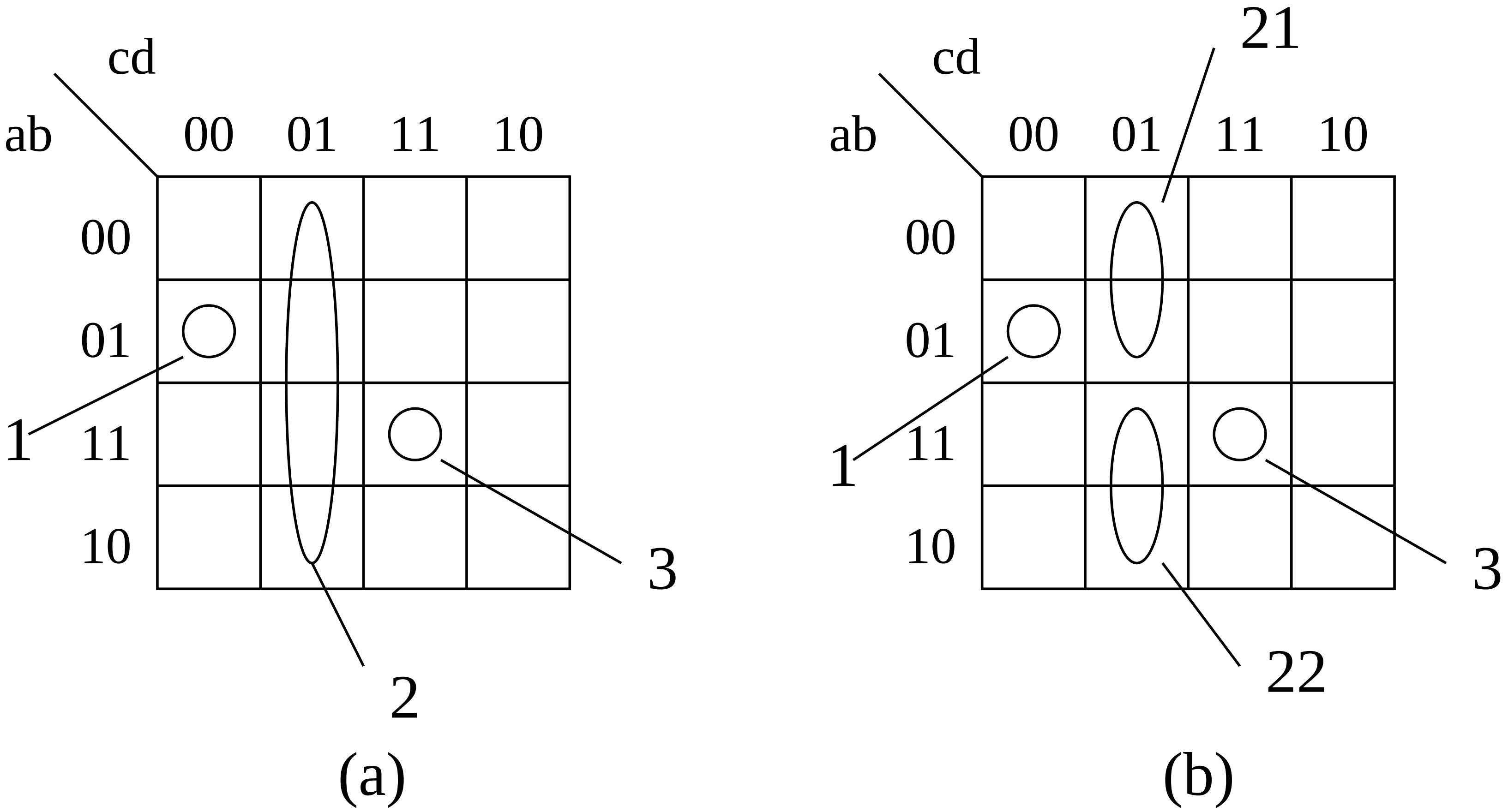
- Cube-splitting operations are performed on the input functions before the graph coloring.
- Essential and secondary essential prime implicants and CDEC-implicants play a role in splitting.
- A heuristic algorithm splits the given input cubes and generates a *set of split cubes*.
- This new set of cubes,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

SPLITTING OF CUBES

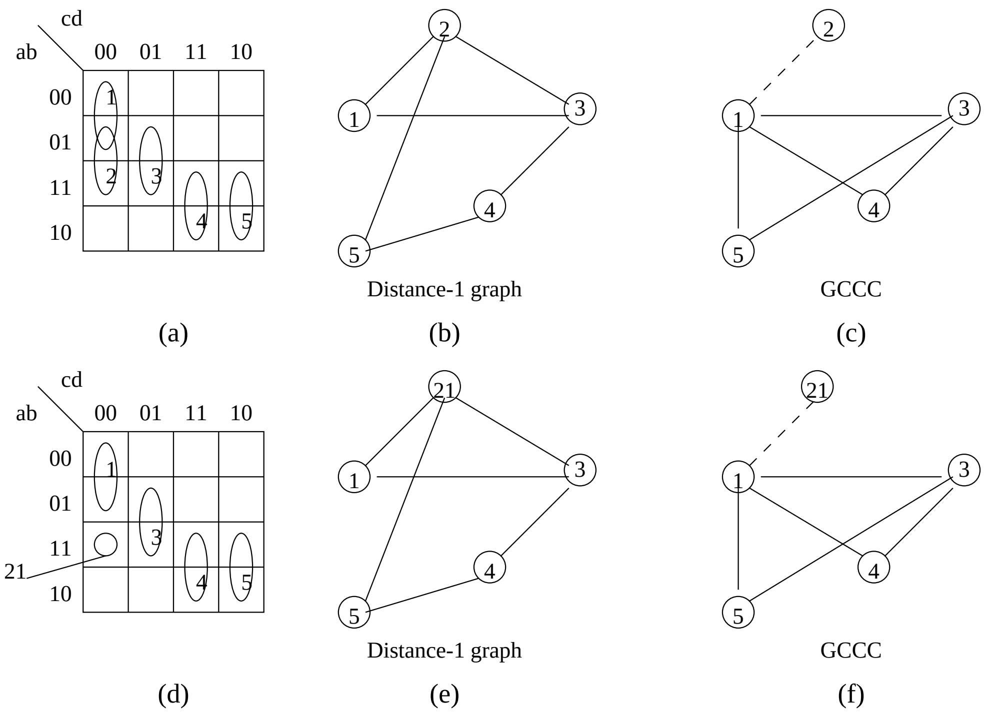
Definition. Distance-1 graph of cubes is a graph with nodes corresponding to cubes and edges corresponding to pairs of cubes that have Hamming distance not greater than 1.

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

EXAMPLE OF CUBE SPLITTING



ALGORITHM FOR CUBE SPLITTING



(a) initial cubes, (b) initial distance-1 graph, (c) initial GCCC, (d) result of sharping, (g) result of Initial-split, (d) - (f) - the state of the graphs when Algorithm 2 terminates.

ALGORITHM FOR CUBE SPLITTING. II

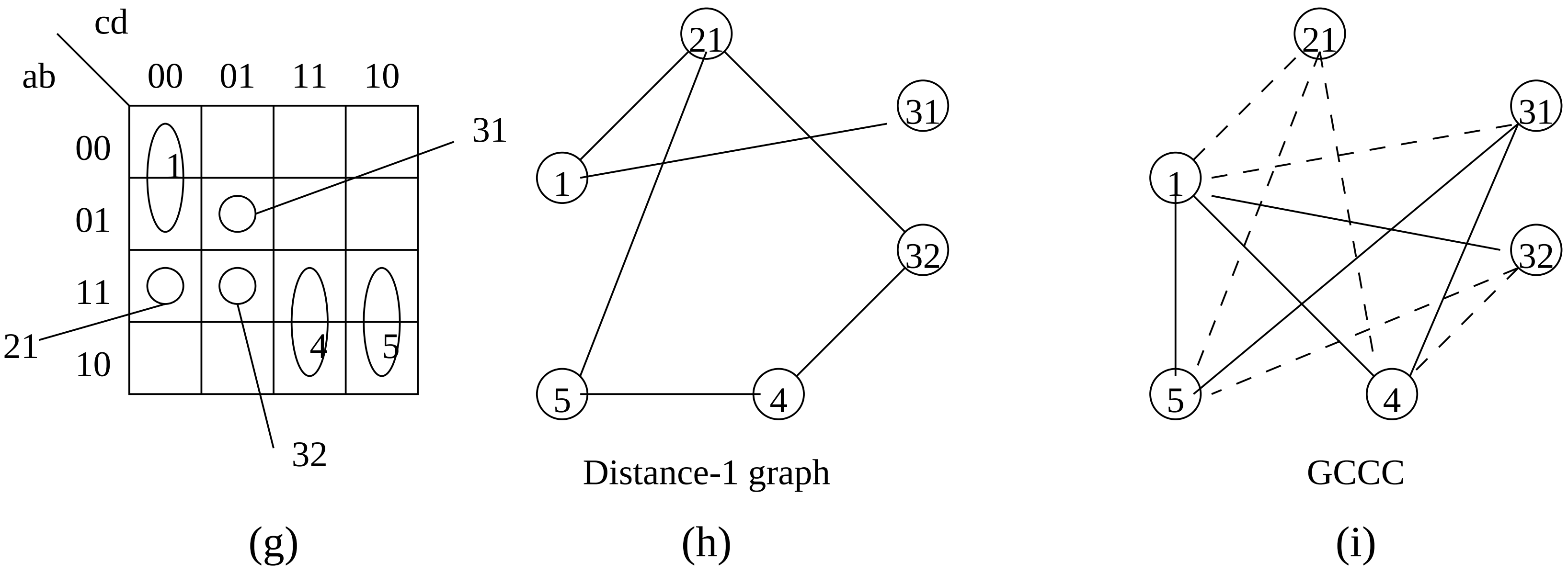


Figure 3: *(g) - (i) - the state of the graphs when Splitting Algorithm terminates. Its Terminating Condition is satisfied.*

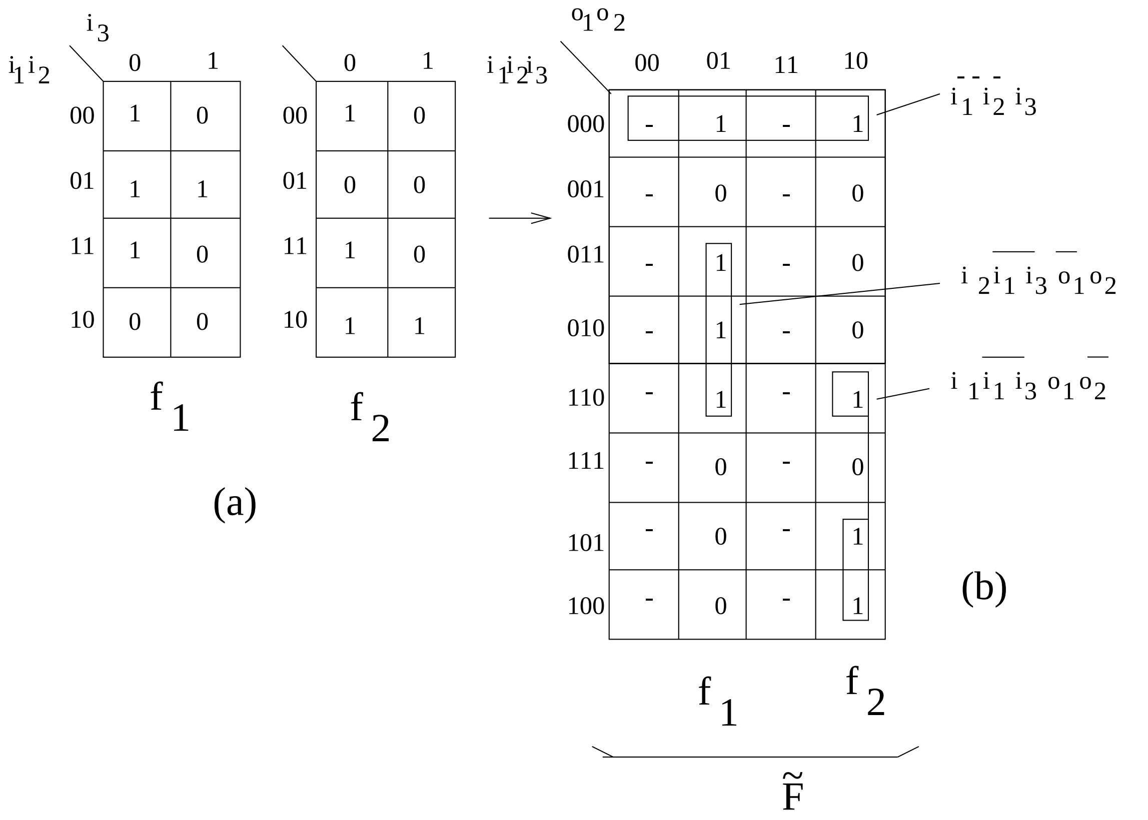
MULTI-OUTPUT TO SINGLE-OUTPUT TRANSFORMATION

- Function represented by $ON(f_i)$ $OFF(f_i)$.
- Multi-output to single-output transformation.
- *Muller transformation* transforms a multi-output function with *m* input and *n* output variables to a single-output function with *m*+*n* input variables.
- This method is not practical for functions represented by ON and DC sets (such as in Espresso), because it creates as many as $m \cdot (2^n - n)$ additional DC-minterms.
- We use ON and OFF sets.

MULTI-OUTPUT TO SINGLE-OUTPUT TRANSFORMATION

- Because only some combinations of new variables are used, the new function is very strongly unspecified.
- This approach is very efficient for data represented with ON, OFF covers, which contradicts the previous opinion that Muller method is impractical.
- In our representation, the algorithm with Muller method as a preprocessor is very fast.
- This approach can be used to **extend arbitrary single-output minimizer that uses ON,OFF representation for (in)complete functions into a multi-output minimizer**.

EXAMPLE OF MULLER TRANSFORM



$$f_1 = \bar{i}_1\bar{i}_2\bar{i}_3 + i_2\overline{(i_1 i_3)}, \qquad f_2 = \bar{i}_1\bar{i}_2\bar{i}_3 + i_1\overline{(i_2 i_3)}$$

EXPERIMENTAL RESULTS OF SOCMIN. GOALS OF EXPERIMENTS

- SOCMIN was tested on MCNC, Cypress, and Machine Learning benchmarks.
- Contrary to MCNC and Cypress benchmarks, the Machine Learning benchmarks from U.C. Irvine have a very large number of don't cares (we binary-encoded the multiple-valued variables).
- We verified all our results with SIS verifier.

The **goals of experiments** were to answer the questions:

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

EXPERIMENTS WITH MULTI-OUTPUT FUNCTIONS

- Even with cube-splitting preprocessing our algorithm is very fast.
- In all cases 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.

HOW MUCH WE GAIN COMPARING TO ESPRESSO?

- Some functions, *rd73, rd84, vg2,* demonstrate that the CDEC cover is substantially smaller than the SOP cover - in the case of *rd84* there are only 30 CDEC-implicants in the cover found by SOCMIN versus 255 primes in the cover from Espresso.
- For multi-output functions CDEC-covers are better than prime implicant covers.
- SOCMIN produces "high quality" CDEC-covers, and is very fast (the Times were calculated for the last column).

EXPERIMENTAL RESULTS FOR SINGLE-OUTPUT FUNCTIONS

- On single-output functions, the results of the variant using cubes generated by Espresso as the starting point is usually only slightly worse than using Cube Splitting Algorithm.

  (although there are exceptions, like *rd733* where SOCMIN starting from splitted cubes generated a cover with only 13 terms, while when starting from Espresso primes the best CDEC cover had as many as 25 terms).

- In EPLD environment, our original motivation, **even one gate counts** because it could mean the entire chip, so Cube Splitting Algorithm should be always used.

EXPERIMENTAL RESULTS ON MACHINE LEARNING FUNCTIONS

- To verify SOCMIN quality on strongly unspecified functions, it was tested for ML examples with high percentage of don't cares.
- Very good results compared to Espresso have been obtained.
- The numbers of terms, literals, and DFC were compared (DFC is a standard metric used in ML).
- 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.

EXPERIMENTAL RESULTS ON MACHINE LEARNING FUNCTIONS

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

DFC COST METRIC. ADDITIONAL SLIDE.

- DFC was calculated by adding the cardinalities of each of the subfunctions in the decomposition.
- The DFC for arbitrary non-decomposable block is 2^k where k is the number of inputs to the block.
- In "gate-based" minimizers the 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.

MULTI-OUTPUT MCNC BENCHMARKS.

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

Graphs for coloring created from minterms (mnt), cubes from Espresso-minimized SOP, or cube covers calculated by respective minimizers. Column SOCMIN has results calculated using cube splitting. Times are total, given for split cubes.

NUMBERS OF TERMS AND CDEC IMPLICANTS IN COVERS FOR SINGLE-OUTPUT MCNC BENCHMARKS

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

COMPARISON OF ESPRESSO AND SOCMIN ON MACHINE LEARNING BENCHMARKS

Example	ON	Terms		Literals		DFC		Time
		Es	So	Es	So	Es	So	sec
add4	128	2	2	4	4	12	12	0.50
ch177f0	128	2	2	4	4	12	12	0.50
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
intrvl2	128	22	14	110	92	436	380	4.09
kdd1	160	2	2	3	3	8	16	0.03
kdd6	240	4	1	4	4	12	12	0.76
kdd7	175	4	4	8	8	28	28	0.85
maj_gate	93	56	20	280	151	1116	612	10.3
mux8	128	4	4	12	12	44	60	0.57
pal_db_op	160	29	23	151	136	600	600	2.95
parity	128	128	128	1024	1024	4092	4092	22.8
rnd3	134	49	32	306	240	1240	972	7.39
sbtrct2	128	2	2	4	4	12	12	0.48

Comparison of numbers of terms, numbers of literals, and DFC values.

ANALYSIS OF THE ROLE OF DON'T CARES

example	25% don't cares				50% don't cares				90% don't cares			
	Espresso		Socmin		Espresso		Socmin		Espresso		Socmin	
	#t	DFC	#t	DFC	#t	DFC	#t	DFC	#t	DFC	#t	DFC
add0	27	620	6	196	26	668	6	124	13	408	3	44
ch8f0	38	816	32	672	36	860	6	184	18	548	3	68
grt_than	28	628	6	188	31	776	10	264	12	372	2	28
intrvl2	27	612	13	468	27	692	9	248	10	300	3	48
kdd1	24	492	2	12	23	544	2	16	11	324	2	20
maj_gate	44	1056	12	456	29	776	10	332	10	308	2	52
mux8	24	516	4	56	21	512	4	68	7	212	1	0
pal_dbl_op	41	992	40	952	35	936	11	352	12	364	3	36
pal_op	42	1108	39	992	34	960	15	568	10	304	2	48
parity	94	3004	20	596	63	2012	14	544	8	252	2	44
rnd1	46	1228	44	1056	36	1004	18	824	12	372	4	60
sbtrct1	43	1160	15	536	31	868	9	340	10	312	0	0

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

CONCLUSIONS. I

1. **Application for CY7C361 PLD.**
   - Saving only one CDEC gate is important (the well-known "fit/no-fit problem").
   - Our algorithm solves this problem.
   - Although we cannot clain that our algorithm is exact, **we did not find any single case that the solution was not the exactly minimum one.**
2. New concept of CDEC PLA.
   - Its practical usefulness to reduce the number of terms has been demonstrated.
   - The respective multi-output algorithms have been developed and compared to known PLA tools.
   - Several variants of high quality algorithms for a general problem of multi-output CDEC minimization have been implemented.
   - They give very good results, proving the usefulness of the **CDEC PLA concept**.
3. The usefulness of **Muller transform** for ON,OFF representations was demonstrated.

CONCLUSIONS. CONDITIONAL GRAPH COLORING

1. New general approach to solve covering problems.
2. The algorithm gives very good solutions, better than Espresso, for multi-output strongly unspecified functions.
3. This has several applications:
   - PLA minimization,
   - Column Minimization Problem in Curtis-like decomposition of Boolean and Multi-valued Relations.
   - controller design,
   - Machine Learning,
   - Knowledge Discovery from Databases,
   - functions created in multi-level or multi-output optimization problems (such as Muller transform).