An Efficient Algorithm for Constrained Encoding and its Applications

Chuan-Jin Shi, *Student Member, IEEE*, and Janusz A. Brzozowski, *Member, IEEE*

***Abstract*—In this paper, an efficient algorithm and its implementation ENCORE are presented for finding approximate solutions to dichotomy-based constrained encoding, a problem fundamental to the synthesis of combinational logic circuits, and synchronous and asynchronous sequential circuits. ENCORE adopts a greedy strategy to find an encoding bit by bit, and then uses an iterative method to improve the solution quality. The novelty of our algorithm lies mainly in a linear-time heuristic to select each individual bit; this problem was previously solved in quadratic time. ENCORE has been applied to a variety of practical problem instances. For a number of examples found in the literature on the synthesis of asynchronous sequential machines, ENCORE consistently obtains optimal or near-optimal results. For the optimum state assignment of the MCNC FSM benchmarks, ENCORE generates the same or even shorter encoding lengths than the programs KISS, NOVA and DIET, but takes much less CPU time. It is demonstrated for the first time that PLA implementations of synchronous FSMs using dichotomy constraints compare very favorably with respect to area with those based on traditional group constraints.**

***Index Terms*—constrained encoding, state assignment, logic synthesis asynchronous circuits, PLA decomposition, local search heuristics.**

I. Introduction

Constrained encoding is an important problem arising in many aspects of the synthesis of combinational and sequential logic circuits. Given a set $S = \{s_1, \cdots, s_m\}$ of m *states*, the (*complete*) *constrained encoding* problem is to find an encoding α of S into a set $\{\alpha(s_1), \cdots, \alpha(s_m)\}$ of m binary k-tuples (k-bit vectors), in such a way that all the dichotomy constraints (defined below) are satisfied and k is minimized. A (*partial*) *dichotomy constraint* requires that a subset P of S be distinguished from a disjoint subset Q of S by at least one bit, i.e., that bit must have the value 0 for all the states in P and 1 for all the states in Q, or vice versa. If Q is empty, we have the special case of *unary constraint* requiring that a subset P of S must be identified by at least one bit b of the k-tuples, in the sense that the value of b should be the same for all the states in P. A variation, called the *partial constrained encoding* problem, aims at maximizing the number of constraints that are satisfied using a fixed number of bits. For example, consider a unary constraint ($\{s_1, s_2, s_4\}$), and four dichotomy constraints ($\{s_1, s_3\}, \{s_2\}$), ($\{s_1, s_3\}, \{s_4\}$), ($\{s_3, s_4\}, \{s_1\}$), and ($\{s_3, s_4\}, \{s_2\}$), on the set $S = \{s_1, \cdots, s_4\}$. Table I(a) shows a minimum-length encoding satisfying all the constraints. Table I(b) gives a two-bit encoding satisfying the largest number (4) of constraints.

TABLE I
Examples of Encodings

s	$\alpha(s)$			s	$\alpha(s)$	
s_1	1	0	0	s_1	1	0
s_2	1	0	1	s_2	1	1
s_3	0	1	0	s_3	0	0
s_4	1	1	1	s_4	0	1
	(a)				(b)	

The constrained encoding problem was first formulated by Tracey [18] for critical-race-free state assignments of asynchronous finite state machines (FSM's). Unger [19] pointed out that, for certain kinds of FSM's, the problem of obtaining an asynchronous implementation, where correctness is independent of the presence of arbitrary gate and wire delays can be reduced to the problem of constrained encoding. Recent studies indicate that the problem of encoding states of FSM's to have a minimum PLA implementation is related to the partial constrained encoding problem [21], [23].

The search for efficient solutions for the constrained encoding problem was pioneered by Tracey [18]. He proposed a procedure, similar to Boolean logic minimization, which consists of two basic steps: First, construct all maximal compatible sets of dichotomy constraints; each such set can be satisfied by one bit assignment, called a *prime bit assignment*. Second, find a minimal number of prime bit assignments to cover all the given dichotomy constraints: this problem is known as the covering problem. This prime-covering method gives an exact solution to the complete constrained encoding problem. However, the number of prime bit assignments may be exponential in the number of states, and the covering problem is NP-complete [8]. In practice, the process described above has been approximated using various heuristics [20], [23]. In addition, it is not clear how to apply the prime-covering

Manuscript received March 29, 1993. This work was supported by the Natural Sciences and Engineering Research Council of Canada under Grant OGP0000871, and by a grant from the Information Technology Research Centre of Ontario. A preliminary version of this paper appeared in Proc. Euro-DAC'92, Hamburg, Germany, September 7-10, 1992, pp. 266-271. This paper was recommended by Associate Editor L. Trevillyan.

The authors are with the Department of Computer Science, University of Waterloo, Waterloo, ON, Canada N2L 3G1.

IEEE Log Number 9212345.

approach to partial constrained encoding. As a consequence, for the optimum state assignment for synchronous sequential machines, the dichotomy-based approaches (such as DIET [23]) have not been as successful as classical approaches (such as KISS [9], CREAM [10], NOVA [21], and [13]), which are based on group constraints. A *group constraint* specifies that a set of states must be encoded in a neighbourhood or a face in the Boolean space.

This paper uses a variant of the prime-covering approach to find good approximate solutions for complete and partial constrained encoding in a unified manner. It replaces the generation of prime bit assignments and the solution of the covering problem by a single step. A single bit of an encoding is generated so as to satisfy as many constraints as possible; this is defined as the *optimal bit generation* problem. To find a complete encoding, we repeat this bit generation process until all the constraints are satisfied. To find a bounded-length encoding, we repeat the bit generation process until the number of bits generated reaches the bound. To derive an efficient algorithm for optimal bit generation, we make a new observation which is made possible by extending constrained encoding to include unary constraints, i.e., allowing one block of a dichotomy constraint to be empty. Noticing that the well-known two-way network partitioning problem is a special case of the partial constrained encoding problem, we are able to generalize the successful heuristic of Fiduccia and Mattheyses [7] for network partitioning to optimal bit generation. By doing this, we achieve a fast approximation algorithm with computational cost linearly proportional to the problem size.

This paper is structured as follows: Section II introduces some basic definitions and the problem formulation. The basic idea of the encoding algorithm is illustrated in Section III. Section IV presents the encoding algorithm, along with its time complexity analysis. Section V describes some extensions and improvements. A description of various applications to logic synthesis is provided in Section VI. Section VII reports some experimental results. Section VIII concludes the paper.

II. Problem Formulation

In this section we give a mathematical formulation of constrained encoding. A *constraint* c on a set $S = \{s_1, \cdots, s_m\}$ is a pair $c = (c^+, c^-)$ of disjoint subsets (called *blocks) of* S. We may distinguish two types of constraints: A (*partial*) *dichotomy constraint* consists of two nonempty blocks. A *unary constraint* is a constraint with one empty block.

Let $B = \{-1, 1\}$ represent the two logic values. Conventionally, $\{0, 1\}$ is used, but the use of $\{-1, 1\}$ will considerably simplify our notation. Given a set $S = \{s_1, \cdots, s_m\}$ of $m > 0$ *states* and an integer $k > 0$, a *binary encoding* (or simply an *encoding*) $\mathbf{a}$ of S is a mapping $\mathbf{a}$: $S \rightarrow B^k$. Note that $\mathbf{a}$ need not be one to one. We may think of the encoding as a matrix $\mathbf{A}$: The ith row of the matrix represents the *word* assigned by $\mathbf{a}$ to state s_i, and the jth column represents *bit* j of the encoding. We use α to refer to a particular column of $\mathbf{A}$; such a column is called a *bit assignment*, and can be interpreted as a mapping α: $S \rightarrow B$. We denote by $\alpha_1, \cdots, \alpha_m$ the components of bit assignment α.

With a slight abuse of notation, we say that state s_i is *contained* in constraint c and write $s_i \in c$ iff $s_i \in c^+ \cup c^-$. We use $|c|$ to denote the number of states contained in constraint c. A set $C = \{c_1, \cdots, c_n\}$ of n constraints can be described by the *constraint matrix* $\mathbf{C} = (c_{ij})_{m \times n}$, where

$$c_{ij} = \begin{cases} 1 & \text{if } s_i \in c_j^+, \\ -1 & \text{if } s_i \in c_j^-, \\ 0 & \text{if } s_i \notin c_j. \end{cases}$$

We use C_i to denote the subset of constraints in C containing s_i, and we let $|C_i|$ be its cardinality.

A bit assignment $\mathbf{a}$: $S \rightarrow B$ is said to *satisfy* a constraint $c = (c^+, c^-)$ iff there exists a value $b \in B$ such that for all $s \in c^+$, $\alpha(s) = b$, and for all $s \in c^-$, $\alpha(s) = \bar{b}$, where $\bar{b}$ is the complement of b. An encoding $\mathbf{a}$: $S \rightarrow B^k$ is said to *satisfy* a constraint $c = (c^+, c^-)$ iff at least one bit assignment of $\mathbf{a}$ satisfies c.

To illustrate these definitions, let $S = \{1, \cdots, 6\}$ and consider bit assignments α, β, γ, and ϵ shown in Table II and constraints c_1, c_2, c_3, and c_4 defined below.

- $c_1 = (\emptyset, \emptyset)$ and $c_2 = (\{3\}, \emptyset)$. Any bit assignment satisfies c_1 and c_2. Therefore constraint $(\emptyset, \emptyset)$ and constraints with one empty block and one one-state block are *trivial* constraints, and will be excluded.
- $c_3 = (\{2, 3, 5\}, \emptyset)$. Bit assignments α, γ and ϵ satisfy c_3, but β does not.
- $c_4 = (\{1, 2, 5\}, \{4, 6\})$. Bit assignments α and β satisfy c_4, but γ and ϵ do not.

The encoding composed of bit assignments α, β, γ and ϵ satisfies all four constraints.

The (*complete*) *constrained encoding problem* is defined as follows: Given a set S of m states, and a set C of n constraints on S, find an encoding $\mathbf{a}$ of S with minimum k, such that $\mathbf{a}$ satisfies each constraint $c \in C$. A variation of this problem, the *partial constrained encoding problem*, is as follows: Given a set S of m states, a set C of n constraints on S, and an integer h, find an encoding $\mathbf{a}$ of S with $k = h$ such that $\mathbf{a}$ satisfies as many constraints of C as possible. If $h = 1$, this problem is called the *optimal bit generation* problem.

In describing the running time of an algorithm on a given instance of the constrained encoding problem, we measure the *size of the input* in terms of $p = \Sigma_{j=1}^{n} |c_j|$. Clearly, the problem size is the number of nonzero entries in the constraint matrix.

III. An Overview of the Encoding Algorithm

In this section, we describe our algorithm informally. Consider the following example. Let $S = \{s_1, \cdots, s_5\}$ and $C = \{c_1, \cdots, c_4\}$, where C is described by the

TABLE II
EXAMPLES OF ASSIGNMENTS

s	$\alpha(s)$	$\beta(s)$	$\gamma(s)$	$\epsilon(s)$
1	−1	1	−1	1
2	−1	1	1	1
3	−1	−1	1	1
4	1	−1	−1	1
5	−1	1	1	1
6	1	−1	−1	1

following constraint matrix:

$$\mathbf{C} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & -1 \\ 0 & 1 & -1 & -1 \end{pmatrix}.$$

We want to find a minimum-length encoding that satisfies $\mathbf{C}$. We will do this by constructing a sequence $\alpha_1, \alpha_2, \cdots$ of bit assignments. The general strategy is to first find an α that satisfies as many constraints as possible. The process is then repeated with the as-yet-unsatisfied constraints until all the constraints are satisfied.

To find a bit assignment, we arbitrarily choose the vector $\alpha^{(0)} = (1, 1, 1, 1, 1)^T$—$T$ denotes the transpose—as the initial bit assignment or "seed." This vector will then be modified in a series of "moves;" each time only one component of α is changed so as to satisfy as many of the constraints as possible.

If we use the value 0 to denote "don't cares," then we may speak of "ternary" ($\{-1, 0, 1\}$) bit assignments. Any nontrivial constraint c_j is satisfied by two "ternary" bit assignments: one is equal to the jth column of $\mathbf{C}$, and the other is the negation of that column. We denote these assignments by c_j and $-c_j$, respectively.

In order to decide which component of the present bit assignment α should be changed to get closer to a solution, we define the *direction matrix* $\mathbf{\Delta} = (\delta_{ij})_{m \times n}$, where $\delta_{ij} = \alpha_i c_{ij}$. If α_i agrees with c_{ij}—the ith component of c_j—then δ_{ij} is equal to 1; if α_i disagrees with c_{ij}, then δ_{ij} is equal to -1; finally, if s_i is not in c_j, then $c_{ij} = 0$ and δ_{ij} is also 0. Therefore the number of -1 entries in each column of Δ reflects how far α is from bit assignment c_j; we denote this number by d_j^+ and call it the *distance* from α to c_j. Similarly, the number d_j^- of 1 entries in column j of Δ is the distance from α to $-c_j$. Each entry in $\mathbf{\Delta}$ has the following meaning:

$$\delta_{ij} = \begin{cases} 1, & \text{if changing } \alpha_i \text{ takes } \alpha \text{ away from } c_j \\ & \quad \text{(closer to } -c_j\text{) by 1,} \\ -1, & \text{if changing } \alpha_i \text{ takes } \alpha \text{ closer to } c_j \\ & \quad \text{(away from } -c_j\text{) by 1,} \\ 1, & \text{otherwise.} \end{cases}$$

The shortest distance, called d_j, from α to an assignment that satisfies constraint c_j is $d_j = \min \{d_j^+, d_j^-\}$. If $d_j \neq 0$, then constraint c_j is not satisfied by α. We will denote by d^+ the distance vector $(d_1^+, \cdots, d_m^+)$, and treat d^- and d similarly. The number u of unsatisfied constraints is the number of nonzero components of d.

Initialization: The direction matrix for initial bit assignmnet $\alpha^{(0)}$ is simply $\mathbf{\Delta}^{(0)} = \mathbf{C}$. The calculation of d^+, d^-, and d, and u is illustrated below.

$$\overset{\alpha^{(0)}}{\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}} \quad \mathbf{\Delta}^{(0)} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & -1 \\ 0 & 1 & -1 & -1 \end{pmatrix}$$

$$\begin{aligned} d^+ &= (1 \;\; 0 \;\; 1 \;\; 2) \\ d^- &= (1 \;\; 3 \;\; 2 \;\; 2) \\ d &= (1 \;\; 0 \;\; 1 \;\; 2) \\ u &= 3. \end{aligned}$$

Since $u \neq 0$, we consider which component of α should be changed to get as close to a solution as possile. For this purpose we introduce the *gain matrix* $\mathbf{G} = (g_{ij})_{m \times n}$ where

$$g_{ij} = \begin{cases} 1, & \text{if the change of } \alpha_i \text{ changes } c_j \text{ from} \\ & \quad \text{unsatisfied to satisfied,} \\ -1, & \text{if the change of } \alpha_i \text{ changes } c_j \text{ from} \\ & \quad \text{satisfied to unsatisfied,} \\ 0, & \text{otherwise.} \end{cases}$$

Matrix $\mathbf{G}$ can be obtained by inspection of d and $\mathbf{\Delta}$. There are three situations: If $d_j = 0$—as in the case of d_2—then constraint c_j is satisfied by α. If we change any component of α corresponding to a state in c_j, the number u of unsatisfied constraints will increase by 1. Therefore $g_{12} = g_{22} = g_{52} = -1$. If $d_j = 1$— as in the case of d_3—there exists a component of α (here α_5) which, if changed, will cause c_j to become satisfied. Therefore the corresponding entry (g_{53}) in $\mathbf{G}$ is 1. In the special case where $d_j = 1$ and $|c_j| = 2$—as in the case of c_1—both entries are equal to 1. Thus $g_{21} = g_{41} = 1$. If $d_j > 1$—as in the case of d_4—constraint c_j will remain unsatisfied no matter which component of α changes. Thus all the entries in the corresponding column (here column 4) are 0. The above calculations result in the matrix $\mathbf{G}^{(0)}$ shown below. The sum of the entries in row i of $\mathbf{G}$ is the total gain obtained by complementing α_i, which forms a *gain vector* γ. In our example, the fourth component has the largest gain; hence, we select it as the component to be changed.

$$\mathbf{G}^{(0)} = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 \end{pmatrix} \overset{\gamma^{(0)}}{\begin{pmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}}$$

select α_4.

Move 1: We change the fourth component of $\alpha^{(0)}$, and the new bit assignment is $\alpha^{(1)} = (1, 1, 1, -1, 1)^T$. This is the first *move*. The new direction matrix $\Delta^{(1)}$ is the same as Δ except that the 4th row is the negation of the original row. The fourth component of α is "locked" after the move, in the sense that it will not be changed again during the search for the first bit assignment. Hence we do not need to check the gain entries corresponding to this component; such entries will be denoted by X. All of the calculations connected with Move 1 are compactly summarized below.

$$\begin{array}{c}\alpha^{(1)}\\ \begin{pmatrix}1\\1\\1\\-1\\1\end{pmatrix}\end{array} \quad \Delta^{(1)} = \begin{pmatrix}0&1&1&1\\1&1&1&0\\0&0&0&1\\1&0&0&1\\0&1&1&1\end{pmatrix}$$

$$\begin{aligned} d^+ &= (0\ \ 0\ \ 1\ \ 2)\\ d^- &= (2\ \ 3\ \ 2\ \ 3)\\ d &= (0\ \ 0\ \ 1\ \ 1)\\ u &= 2.\end{aligned}$$

$$\mathbf{G}^{(1)} = \begin{pmatrix}0&-1&0&0\\-1&-1&0&0\\0&0&0&0\\-1&0&0&0\\0&-1&1&1\end{pmatrix} \begin{array}{c}\gamma^{(1)}\\ \begin{pmatrix}-1\\-2\\0\\X\\1\end{pmatrix}\end{array}$$

select α_5

Move 2: According to $\gamma^{(1)}$, we change α_5. The results are as follows:

$$\begin{array}{c}\alpha^{(2)}\\ \begin{pmatrix}1\\1\\1\\-1\\-1\end{pmatrix}\end{array} \quad \Delta^{(2)} = \begin{pmatrix}0&1&1&1\\1&1&1&0\\0&0&0&1\\1&0&0&1\\0&-1&1&1\end{pmatrix}$$

$$\begin{aligned} d^+ &= (0\ \ 1\ \ 0\ \ 0)\\ d^- &= (2\ \ 2\ \ 3\ \ 4)\\ d &= (0\ \ 1\ \ 0\ \ 0)\\ u &= 1.\end{aligned}$$

$$\mathbf{G}^{(2)} = \begin{pmatrix}0&0&-1&-1\\-1&0&-1&0\\0&0&0&-1\\-1&0&0&-1\\0&1&-1&-1\end{pmatrix} \begin{array}{c}\gamma^{(2)}\\ \begin{pmatrix}-2\\-2\\-1\\X\\X\end{pmatrix}\end{array}$$

Stop

All the entries in the new gain vector $\gamma^{(2)}$ are either X or negative. This means that the corresponding components of α are either locked or, if changed, will cause more constraints to be unsatisfied. Our strategy, therefore, is to terminate the search for the first bit assignment. Thus we obtain $\alpha = (1, 1, 1, -1, -1)^T$, which satisfies constraints c_1, c_3 and c_4.

We repeat the search above for the second bit assignment, but with set $C = \{c_2\}$. We obtain our second bit assignment $(1, 1, 1, 1, 1)^T$. Because all the given constraints are satisfied by the two bit assignments, we have found a solution of the encoding problem.

In the description above, the direction and gain matrices are introduced for descriptive convenience. What we really need are the distance and gain vectors. We can divide the computational cost associated with each move into three parts:

- *Task 1:* The computation of the distance vectors.
- *Task 2:* The calculation of the gain vector.
- *Task 3:* The selection of the component in the gain vector that has the maximum value.

In the next section, we will see how this idea can be implemented so that its worst-case computational cost is $O(p)$, where p is the number of nonzero elements in the constraint matrix.

IV. Efficient Implementation of the Encoding Algorithm

We now describe an efficient implementation of the basic ideas of the previous section. Efficiency is achieved by using suitable data structures for Tasks 1 and 3, and an incremental approach for Task 2, so that all three tasks are performed in $O(p)$ time.

A. Calculation of Distance Vectors

We use sparse matrix techniques to store the constraint matrix $\mathbf{C}$. We maintain an array of constraints, in which each entry is a linked list of pointers to states in that constraint. We also keep an array of states, in which each entry is a linked list of pointers to constraints involving the state. These two arrays permit efficient traversal of nonzero entries by row or by column. We also keep track of the values of d_j^+ and d_j^- for each constraint c_j. After changing α_i, we update the distance values as follows:

$$d_j^+ = d_j^+(\text{old}) + \delta_{ij},$$

$$d_j^- = d_j^-(\text{old}) - \delta_{ij}.$$

The number of operations needed for computing the new distances is thus $|C_i|$. The total number of operations needed to maintain distance vectors for generating one bit assignment is $\Sigma_{i=1}^{m} |C_i| = p$. The initialization of d^+ and d^- takes p operations. Therefore the total cost for Task 1 is $2p$, i.e., $O(p)$.

Proposition 1: Using the sparse matrix data structures, the calculation of distance vectors in the entire process of bit generation takes $O(p)$ time.

B. Initialization of Gain Vectors

Now we consider how to calculate the gain vector for a given bit assignment α. All distances can be obtained in $O(p)$ time. Only constraints with the shortest distances of 0 or 1 contribute to the gain vector; such constraints are said to be *α-sensitive.* The calculation of the gain vector checks each constraint to see if it is α-sensitive. If so, all the states in that constraint may be checked to determine whether to add 1, remove 1, or do nothing for that corresponding entry. In the worst case, all the constraints are α-sensitive, and all the states in each constraint need to be checked. Therefore $O(p)$ operations are needed to build up the gain vector for a given α.

Proposition 2: Computing the gain vector for a given bit assignment takes $O(p)$ time.

C. Incremental Gain Updating

Knowing the gain vector for α, we calculate the gain vector for the new α obtained by changing α_i. An efficient approach is to perform incremental updating, i.e., modify only that part of the gain vector that is affected by the move.

To illustrate this idea, we refer to Move 1 in the generation of Bit 1 in our introductory example. Both the distance and gain vectors for $\alpha^{(0)}$ are known. We change α_4, and all the nonzero entries in the fourth row of $\mathbf{\Delta}^{(0)}$ (corresponding to s_4) are negated. These entries are δ_{41} and δ_{44}. Therefore, only the distances for constraints in C_4 are changed (here $C_4 = \{c_1, c_4\}$). As a consequence, only the first and the fourth columns of $\mathbf{G}$ may be changed. Since γ_i is equal to the sum of the entries in the ith row of $\mathbf{G}$, the new γ can be obtained from the old γ by first subtracting the first and the fourth columns of $\mathbf{G}^{(0)}$ and then adding the first and the fourth columns of $\mathbf{G}^{(1)}$. Thus, the calculation of contributions to the gain vector is required only for the constraints in C_i.

Gain updating is required only when a constraint in C_i is sensitive in the present or the previous moves. Due to the fact that each component is locked after the move, a constraint c_j can be sensitive ($d_j = 1$ or $d_j = 0$) during the search for one bit assignment only a constant number of times. The reason is as follows: A moved component either agrees or disagrees with c_j. Consider the first case. Since the component is locked after the move, it will disagree with c_j forever. This means that d_j^+ will be greater than 0 forever. Similarly, for the second case, we can conclude that d_j^- will be greater than 0 forever. Thus the number of times that d_j is 0 is bounded. If there are two moves involving two components that originally agree with c_j, then, after the move, the two components will disagree with c_j forever, i.e., $d_j^+ > 1$. Similar reasoning shows that the number of times that d_j is 1 is also bounded. Formally, we have the following result.

Proposition 3: For each constraint, gain updating is required only a constant number of times during the entire process of bit generation.

A detailed case analysis reveals that three gain updatings are sufficient [15]. Since the number of updatings for each constraint is bounded, we have the following proposition.

Proposition 4: For each constraint, all gain updating takes $O(p)$ time during one bit generation.

D. Data Structure for Gain Vectors

Now we describe a data structure, denoted by $\mathcal{B}$, for the gain vector. It shall support the following operations:

- INSERT($\mathcal{B}$, i, g_i) inserts a component i with g_i in $\mathcal{B}$.
- DELETE($\mathcal{B}$, i) deletes component i from $\mathcal{B}$.
- UPDATE($\mathcal{B}$, i, f) updates g_i by f, i.e., lets $g_i = g_i + f$, and moves it to the appropriate place in $\mathcal{B}$.
- MAX($\mathcal{B}$) returns the component of α with maximum gain, or NIL if $\mathcal{B}$ is empty.

We use a *bucket list.* The range of the bucket list goes from $-e$ to e, where $e = \max \{|C_i|, 1 \leq i \leq m\}$. The jth entry of the bucket list contains a doubly-linked list of unlocked components with gain currently equal to j. An additional array INDEX is used to maintain pointers for direct access to each component in the bucket list. Whenever a component is locked, we remove it from the bucket list and set the corresponding INDEX to NIL.A MAXGAIN pointer is maintained to keep track of the bucket having a component of highest gain. This pointer is updated by decrementing it whenever its bucket is found to be empty, and resetting it to a higher bucket whenever a component moves to a bucket above MAXGAIN. With the bucket-list data structure, all the operations above except MAX($\mathcal{B}$) take $O(1)$ time.

Proposition 5: Operation MAX takes $O(P)$ time in total in the process of bit generation.

Proof: Operation MAX is performed at most n times during the generation of one bit assignment. Since this is done by accessing pointer MAXGAIN, it is sufficient to examine how much work is needed to maintain MAXGAIN. Pointer MAXGAIN may be affected by INSERT, DELETE and UPDATE. Whenever a component moves to a bucket above MAXGAIN, MAXGAIN is simply reset to a higher bucket. Operation INSERT is invoked m times; therefore $O(p)$ time is needed for maintaining MAXGAIN due to this operation. Whenever a bucket of maximum gain is found to be empty, we need to decrement MAXGAIN until we find the next non-empty bucket. This may happen when operation DELETE is invoked, or when UPDATE is needed to decrease the gain ($f < 0$). The number of times DELETE is invoked is at most n, and each time there are at most $2e$ empty buckets, where $e = \max \{ |C_i|, 1 \leq i \leq m\}$. There $O(p)$ time is needed for maintaining MAXGAIN due to DELETE operations. When UPDATE is invoked to decrease the gain, i.e., f is -1, at most one bucket may be found to be empty. When UPDATE is invoked to increase the gain, the current maximum gain is compared with the new gain to determine MAXGAIN. By Proposition 4, the total time needed for maintaining MAXGAIN due to UPDATE is $O(p)$. Therefore, the total time needed for maintaining MAXGAIN is $O(p)$. □

E. Time Complexity of the ENCORE Algorithm

A formal description of the ENCORE algorithm is given in the appendix, where GENERATE_BIT is for generating one bit assignment, and GREEDY_ENCODING refers to the entire encoding algorithm.

Proposition 6: The running time of algorithm GENERATE_BIT is $O(p)$.

Proof: See the appendix.

Now consider the time complexity of GREEDY_ENCODING. Except for the first bit, the set of constraints used in GENERATE_BIT is a subset of C. Therefore, we have the following result:

Theorem 1: The running time of algorithm GREEDY_ENCODING is $O(kp)$, where k is the length of the encoding, and p is the size of the encoding problem.

V. Improvements and Extensions

The greedy strategy used for constructing the entire encoding and the local search strategy used for finding each indivudual bit lead to a good solution of the encoding algorithm; however, they do not guarantee optimality. In this section, we describe techniques used in ENCORE that have been demonstrated effective in improving the quality of constrained encoding. We also describe several extensions to our basic bit generation algorithm in order to handle the output encoding problem.

The first technique is a novel strategy of searching for a global optimum using available information of local minima. It is based on the following observation. Suppose that we have found an encoding with length k. The kth bit assignment of that encoding was introduced to satisfy some non-empty set C' of constraints, which are not satisfied by the first $k - 1$ bit assignments. This set C' thus appeared to be "hard to satisfy" in the present run. Consequently, we start a new run by choosing the first bit assignment so as to satisfy C'. This process is likely to find another distinct local minimum. Experiments have shown that only a few runs are needed to improve the solution. This strategy can be accomplished by assigning a sufficiently large weight to each constraint in C'. Suppose that each constraint c_j in a given set C is associated with an integer weight w_j. Our algorithm and data structures can be used directly, except that the range of the bucket list is now from $-e$ to e, where $e = \max \{\Sigma_{c_j \in C_i} w_j, 1 \leq i \leq m\}$, and the parameter f used in the operation UPDATE($\mathcal{B}$, i, f) is equal to w_j. This enhancement does not increase the time complexity.

The second technique is to impose a "balance criterion" on bit generation. A bit assignment used to satisfy a constraint $c = (c^+, c^-)$ distinguishes states in c^+ from those in c^-. If there are no more constraints, the number of additional bits needed to distinguish states in c^+ (c^-) from each other is $\lceil \log_2 |c^+| \rceil$ ($\lceil \log_2 |c^-| \rceil$; thus, the minimal number of additional bits is $\max \{ \lceil \log_2 |c^+| \rceil, \lceil \log_2 |c^-| \rceil \}$. Hence, in order to minimize the number of encoding bits, it is desirable to have the number of -1's and 1's in a bit assignment balanced. The balance criterion implemented in ENCORE is as follows: Given an integer p, $0 < p \leq m$, as the desirable number of 1's in a bit assignment, and a tolerance $0 \leq r \leq \min \{p, m - p\}$, a bit assignment is said to be *balanced* if $-2r \leq m - 2p + \Sigma_{i=1}^{m} \alpha_i \leq 2r$. When $p = m/2$, we need to have $-2r \leq \Sigma_{i=1}^{m} \alpha_i \leq 2r$; this is the scheme implemented in DIET [23].

In ENCORE, the balance strategy is accomplished by first generating an initial balanced assignment and then maintaining the balance during the process of bit generation. Starting from the initial bit assignment with all components being 1, ENCORE selects a component with maximum gain to change until the balance criterion is satisfied. In the rest of the first pass and also in all the following passes, a component with maximum gain is selected to move only if changing it would not cause imbalance. Otherwise another component with maximum gain or even the second largest gain is selected and checked for the balance criterion. If there are several components having the same largest gain, we select the one which gives the minimum absolute value of $m - 2p + \Sigma_{i=1}^{m} \alpha_i$.

There is a special case for which optimality is guaranteed by GREEDY_ENCODING. The problem is to find a minimum-length encoding for a set S of m states such that each state is assigned a distinct code word. It can be described in our framework of constrained encoding, by a set of $n = (1/2)m(m - 1)$ dichotomies with one state in each block. We need to add this set of *distinct-state* constraints when we handle partial constrained encoding arising from the optimum state assignment problem of synchronous FSM's (See Section VI).

Now we show how our framework can handle the output encoding problem. As shown in [12], modeling of the output encoding problem requires the dominance and disjunctive constraints, in additional to dichotomy constraints. A row i of $\mathbf{A}$ is said to *dominate* another row if, for each bit position in the second row that contains a 1, the corresponding bit position in the first row also contains a 1. Row i of $\mathbf{A}$ is said to be a *disjunction* of rows j and k, if $\alpha_i = \alpha_j \vee \alpha_k$, for each bit. A formulation of the constrained encoding problem resulting from output encoding is as follows [12]: Given a set of dichotomy constraints, a set of dominance constraints, and a set of disjunction constraints, find an encoding with the minimum number of bits such that it satisfies all the dominance, disjunction, and dichotomy constraints.

We can use the same algorithm for this problem, but each bit generated must satisfy all the dominance constraints and all the disjunction constraints. the dominance constraints can be imposed as follows. Initially $\alpha_i = \alpha_j = 1$. When the component of maximum gain is α_i, we check to see the value of α_j. If $\alpha_j = 1$, then we do not change α_i, or say it is an *infeasible* move prohibited by the dominance requirement. So we select the component of the second largest gain, etc. The disjunction constraints can be handled similarly. Initially $\alpha_i = \alpha_j = \alpha_k = 1$. When the component of maximum gain is α_i, we check the values of α_j and α_k. If either α_j or α_k is equal to 1, then we do not change α_i; this is an *infeasible* move prohibited by the disjunctive constraint. When we have changed α_i and α_j, we select α_i as a component to change in the next move, no matter what γ_i is.

The extension above provides a simple way of handling the output encoding problem, while maintaining the same time complexity as the basic bit generation algorithm. The problem size p must now take into account dominance and disjunction constraints. We note that, in order to satisfy these dominance and disjunction constraints, a framework of ordered dichotomies was introduced, which led to even more complicated prime generation and prime covering [12].

VI. Several Applications

Although the formulation of constrained encoding in the state assignment of asynchronous sequential machines was discovered in the 1960's [18], [19], its relation with the optimal state assignment for synchronous sequential machines was understood only very recently [10], [23]. Indeed, despite the huge volume of literature on sequential logic optimization, the problem is still not fully understood. In this section we describe how dichotomy-based constrained encoding relates to correct and economical sequential logic design.

A. Race-Free State Assignment for Asynchronous Machines

The design of sequential logic circuits begins with a behavioral specification, which is often a *state table*, where columns corresponds to inputs, rows to present states, and entries to transitions. Transitions are ordered pairs representing the next state and the current output, respectively. An example of a state table is given in Table III. To find a logic implementation, states are encoded by binary k-tuples. For example, an encoding of (s_1, s_2, s_3, s_4) is (00, 01, 11, 10). This can be viewed as an assignments of two binary *state variables* y_1 and y_2. With such encoding, the transition functions can be described by Boolean logic functions in terms of state variables.

TABLE III
A Flow Table

	x_1x_2			
	00	01	11	10
s_1	s_1, 0	s_2, 0	s_1, 0	s_1, 0
s_2	s_3, 0	s_2, 0	s_2, 0	s_4, 0
s_3	s_3, 0	s_2, 1	s_1, 0	s_3, 0
s_4	s_3, 0	s_4, 1	s_2, 0	s_4, 0

A circuit is said to be *asynchronous*, if it has no clocks [19]. Such a circuit can be constructed directly from the transition functions and uses feedback lines to relate the current state variables and the next state variables. If more than one state variable must change in the course of a transition, the subsequent state of the circuit may depend on the order in which the state variables change, that is, on the variable that wins the *race*. Such a race is *critical* and may lead to a malfunction of the circuit.

Critical races can be avoided by choosing state encoding carefully. It is assumed that only one binary input variable changes at a time, and that the delays in the feedback lines are sufficiently large to let all the circuit changes take place before any state variable can affect the gate inputs. Suppose that, under a given input, if the machine starts in state s_i it should change to state s_j, while if it starts in state s_k it should remain in s_k. If the transition $s_i \rightarrow s_j$ involves a critical race, the circuit may end in state s_k. To avoid this, it is sufficient that one state variable be assigned one value in states s_i and s_j and the opposite value in state s_k. This constraint is represented by a dichotomy ($\{s_i, s_j\}$, $\{s_k\}$). Two transitions are *disjoint* if the corresponding sets of states involved in the transitions are disjoint. In general, the encoding of states should be such that all the states "spanned" by a transition occurring within one column must have one bit differing from the encoding assigned to the states spanned by any disjoint transition in this column. These conditions are known as *Tracey's conditions* [18]. For example, Tracey's conditions for a race-free implementation of Table III are:

- column 00: ($\{s_1\}$, $\{s_2, s_3, s_4\}$)
- column 01: ($\{s_1, s_2, s_3\}$, $\{s_4\}$)
- column 11: ($\{s_1, s_3\}$, $\{s_2, s_4\}$)
- column 10: ($\{s_1\}$, $\{s_3\}$), ($\{s_1\}$, $\{s_2, s_4\}$), ($\{s_3\}$, $\{s_2, s_4\}$)

B. Delay-Free State Assignment for Asynchronous Machines

In general, to avoid critical races and other delay-related timing problems, one has to insert certain delays in the feedback lines. The question arises whether the states of a given FSM can be encoded in such a way that its correctness is independent of the stray delays in the circuit, *without* the insertion of any delays. Such an encoding is called a *delay-free* assignment. It turns out that a delay-free assignment exits if an FSM satisfies certain conditions discovered by Unger, namely if it has no "essential hazards" [20].

Table III has no essential hazards. Single input changes are assumed. In order to produce a delay-free realization, the encoding of states should be such that all the states involved in every possible transition occurring between any two adjacent columns have at least one state variable differing from all the states involved in any other disjoint transition beginning or ending in one of the two adjacent columns. These are called *Unger's conditions* [19]. From Table III, we have:

- column 00 and column 01: $(\{s_1, s_2, s_3\}, \{s_4\})$, $(\{s_1\}, \{s_2, s_3, s_4\})$
- column 01 and column 11: $(\{s_2, s_4\}, \{s_1\})$, $(\{s_1, \{s_2\}, \{s_4\})$
- column 11 and column 10: $(\{s_2, s_4\}, \{s_1, s_3\})$
- column 10 and column 00: $(\{s_3, s_4\}, \{s_1\})$

C. Optimal State Assignment for Synchronous Machines

In *synchronous design*, clocks are used to control each transition so as to avoid critical races and hazards. The major concern for the state assigment of synchronous FSM's is to find a state encoding so as to minimize the cost of implementation. If a PLA is used to implement combinational logic blocks, then the PLA area, which is the main portion of the chip area, is the objective to minimize. The optimal state assignment problem here is to find a state encoding that has a minimum-area two-level logic implementation.

To show how the state assignment problem here can be solved using dichotomy constraints, we consider the FSM of Table IV. We can group together those entries in the state table that have the same next state, and express the next state function as follows.

$$s_1 = x_2'(s_3 + s_4) \tag{1}$$

$$s_2 = x_1 s_3 + x_2(s_2 + s_4) \tag{2}$$

$$s_3 = x_2'(s_1 + s_2) \tag{3}$$

$$s_4 = x_2 s_1 + x_1' x_2 s_3 \tag{4}$$

There exist many such groupings; we select the one with the minimal number of "groups". This is known as symbolic logic minimization. A good tool for this purpose is ESPRESSO-MV [11].

Note that the area of a PLA is determined by the number of binary variables times the number of distinct product terms. If we encode the states in such a way that each group is represented by one Boolean product of the encoding variables $\alpha_1, \cdots, \alpha_k$, then the number of products in the final logic is no larger than the number of "groups". This can be achieved by using dichotomy constraints, as explained below.

Each group is either a single state or a sum of states. A singleton can be expressed directly as a product of the

TABLE IV
A SYNCHRONOUS STATE TABLE

			x_1x_2	
	00	01	11	10
s_1	s_3	s_4	s_4	—
s_2	s_3	s_2	s_2	—
s_3	s_1	s_4	s_2	—
s_4	s_1	s_2	s_2	—

encoding variables $\alpha_1, \cdots, \alpha_k$. For example, if $\alpha(s_1) = (101)$, then s_1 is represented as $\alpha_1\alpha_2\alpha_3$. If only two states appear in a sum, and the code words assigned to those two states are adjacent, it is still straightforward to represent the sum as one product of $\alpha_1, \cdots, \alpha_k$. Consider $s_1 + s_3$, for example, with $\alpha(s_1) = (100)$ and $\alpha(s_3) = (000)$; then $\alpha(s_1) + \alpha(s_3) = (-00)$, i.e., $s_1 + s_3$ can be represented as $\alpha_2'\alpha_3'$. This is the smallest "subcube" that contains the code words assigned to every state in $\{s_1, s_3\}$. Now, suppose $\alpha(s_1) = (101)$ and $\alpha(s_3) = (110)$, i.e., the two code words are not adjacent. If we still take the smallest subcube containing $\alpha(s_1)$ and $\alpha(s_3)$, that is $(1 - -)$, to represent the sum $s_1 + s_3$, it will include not only code words (101) and (110) assigned to s_1 and s_3, but also two additional code words (100) and (111). Such an encoding would be invalid, if (100) and (111) are assigned to states s_2 and s_4. This can be avoided by setting up constraints $(\{s_1, s_3\}, \{s_2\})$ and $(\{s_1, s_3\}, \{s_4\})$. In general, for every sum $s_{i_1} + s_{i_2} + \cdots + s_{i_j}$ we introduce constraints $(\{s_{i_1}, s_{i_2}, \cdots, s_{i_j}\} \{s_l\})$, for all $l \in I$ but $l \neq \{i_1, i_2, \cdots, i_j\}$, where I denotes the integer set ranging from 1 to n. In our example, we thus have constraints $(\{s_3, s_4\}, \{s_1\})$, $(\{s_3, s_4\}, \{s_2\})$, $(\{s_2, s_4\}, \{s_1\})$, $(\{s_2, s_4\}, \{s_3\})$, $(\{s_1, s_2\}, \{s_3\})$, and $(\{s_1, s_2\}, \{s_4\})$.

It should be noted that partial constrained encoding, i.e., bounded-length encoding, may be more relevant than complete constrained encoding. Partial constrained encoding may result in more product terms, but it uses fewer encoding variables. Since the PLA area is related to the product of these two parameters, it is possible that partial encoding yields less PLA area.

D. PLA Decomposition

Another problem that, surprisingly, resembles optimum state assignment is PLA decomposition [4], [5]. To illustrate why PLA decomposition can be solved within the framework of constrained encoding, we consider a PLA with seven primary inputs and two primary outputs, described by the following expressions:

$$y_1 = x_1x_3x_4'x_6 + x_2x_5x_6'x_7 + x_1x_4x_7' + x_1'x_3'x_4x_7' + x_3'x_5'x_6'x_7' \tag{5}$$

$$y_2 = x_2'x_4'x_6 + x_3x_4'x_6 + x_2'x_5x_6'x_7 + x_1'x_3'x_4x_7' + x_1'x_3'x_5'x_6'x_7' + x_2'x_4'x_5'x_7' \tag{6}$$

This PLA has 10 distinct product terms and cannot be further simplified by using logic minimizers such as ESPRESSO [2].

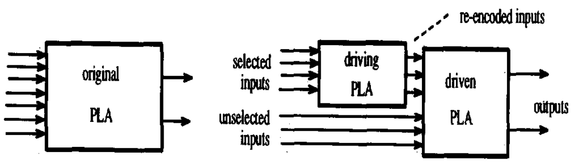


Fig. 1. PLA architecture.

We would like to decompose the given PLA into the configuration of Fig. 1. We assume that the selected subset of inputs is $SI = \{x_4, x_5, x_6, x_7\}$. Five product terms of the selected inputs appear in (5) and (6): $x_4'x_6$, $x_5x_6'x_7$, x_4x_7', $x_5'x_6'x_7'$, and $x_4'x_5'x_7'$. In order to re-encode SI, we first need to make all product terms involving selected inputs *disjoint*. Products x_4x_7' and $x_5'x_6'x_7'$ are not disjoint; neither are $x_5'x_6'x_7'$ and $x_4'x_5'x_7'$. So we expand those terms into minterms. By removing some redundant product terms, the expressions above reduce to

$$y_1 = x_1x_3x_4'x_6 + x_2x_5x_6'x_7 + x_1x_4x_7' + x_1'x_3'x_4x_7' + x_3'x_4'x_5'x_6'x_7' \quad (7)$$

$$y_2 = x_2'x_4'x_6 + x_3x_4'x_6 + x_2'x_5x_6'x_7 + x_1'x_3'x_4x_7' + x_1'x_3'x_4'x_5'x_6'x_7' + x_2'x_4'x_5'x_6'x_7' \quad (8)$$

Now all the product product terms $x_4'x_6$, $x_5x_6'x_7$, x_4x_7', and $x_4'x_5'x_6'x_7'$, of the selected inputs are disjoint. We may view them as four values of a multiple-valued symbolic input variable s, denoted by s_1, s_2, s_3, s_4. Then the above logic expressions with three binary-valued inputs $\{x_1, x_2, x_3\}$, one four-valued input $\{s\}$, and two binary-valued outputs $\{y_1, y_2\}$, can be simplified by multiple-valued symbolic logic minimization. For this example, by using ESPRESSO-MV [11], we find

$$y_1 = x_2s_2 + x_1x_3(s_1 + s_3) + x_3'(s_3 + s_4) \quad (9)$$

$$y_2 = x_2'(s_1 + s_2 + s_4) + x_3s_1 + x_1'x_3'(s_3 + s_4) \quad (10)$$

There are six *symbolic product terms* in these expressions.

We reduce the PLA decomposition to constrained encoding. For this example, we have a total of 5 constraints: $(\{s_1, s_3\}, \{s_2\})$, $(\{s_1, s_3\}, \{s_4\})$, $(\{s_3, s_4\}, \{s_1\})$, $(\{s_3, s_4\}, \{s_2\})$, and $(\{s_1, s_2, s_4\}, \{s_3\})$. It is easily verified that the encoding $\alpha(s_2, s_2, s_3, s_4) = (001, 011, 100, 111)$ is a minimum-length binary encoding satisfying all the constraints. Therefore, we need three binary variables, denoted by x_8, x_9, and x_{10}, to encode the symbolic input variable s. Substituting into (9) and (10), we obtain the re-encoded PLA:

$$y_1 = x_2x_8'x_9x_{10} + x_1x_3x_9' + x_3'x_8 \quad (11)$$

$$y_2 = x_2'x_{10} + x_3x_8'x_9'x_{10} + x_1'x_3'x_8 \quad (12)$$

The driving PLA is expressed as

$$x_8 = x_4x_7' + x_4'x_5'x_6'x_7' \quad (13)$$

$$x_9 = x_5x_6'x_7 + x_4'x_5'x_6'x_7' \quad (14)$$

$$x_{10} = x_4'x_6 + x_5x_6'x_7 + x_4'x_5'x_6'x_7' \quad (15)$$

Note that PLA area is calculated as (2 * (number_of_inputs + number_of_outputs) * number_of_products. Thus the area cost of the original PLA is (2 * 7 + 2) * 10 = 160. The cost of the decomposed PLA, which is the sum of the driving PLA and the driven PLA, is (2 * 4 + 3) * 4 + (2 * (3 + 3) + 2) * 6 = 44 + 84 = 128. Both the original PLA and the decomposed PLA has 10 product terms.

VII. Experimental Results

The proposed encoding algorithm, along with the improvement techniques, has been implemented in a package called ENCORE using the C programming language. In this section we describe some experimental results applied to several sets of problem instances.

The first set of small examples comes from the early literature on the synthesis of asynchronous FSM's. Here the aim is either a race-free [18] or a delay-free implementation [19]. We have written a program to derive the dichotomy constraints from the original flow table specification. ENCORE is then used to find the minimum-length encoding that satisfies all these dichotomy constraints. As summarized in Table V, ENCORE generates encodings with the same lengths as those given by *exact methods* in the literature for all these examples.

The second set of tests consists of 40 industrial examples available from the MCNC benchmarks representing a wide range of FSM's. The raw data can be found in [21]. We have incorporated ENCORE into Berkeley *octtools* to produce PLA realizations from given FSM specifications. We have conducted two groups of experiments. In the first group, we solve the constrained encoding that satisfies all the constraints. We have written a pre-processing program to generate the dichotomy constraints from the group constraints, where group constraints are obtained by running ESPRESSO-MV [2]. We compared ENCORE with several available state assignment programs: KISS [9], NOVA [21], and DIET [23]. The results are summarized in Table VI. For each example tested, the table reports the minimum number *#bits* of bits by unconstrained encoding, the minimum number *#cbits* of bits by constrained encoding, the encoding lengths obtained by KISS, NOVA, DIET, and ENCORE (with and without a balance criterion), and the CPU time used. The reported time for all the tools does not include the time used by ESPRESSO—MV for generating group constraints. The time reported by ENCORE does not include the time used by pre-processing program for generating dichotomy constraints from group constraints, since it is so small that is not measurable. For all the test examples (with the exception of the *keyb* FSM), ENCORE with the balance criterion obtained the shortest length encodings, with only one tenth to one thousandth of the CPU-time used by NOVA and DIET. Note that both ENCORE and DIET work on dichotomy constraints, but DIET uses the prime-covering approach. KISS and NOVA work on the group constraints

TABLE V
SYNTHESIS OF ASYNCHRONOUS FMS's.

FSM	#states	#constraints	#bits	References
fsm1	5	7	3	[19], p. 84
fsm2	6	16	4	[19], p. 97
fsm3	9	19	4	[19], p. 108
fsm4	7	94	4	[19], p. 144
fsm5	5	10	3	[18], Fig. 3
fsm6	6	10	3	[18], Fig. 4

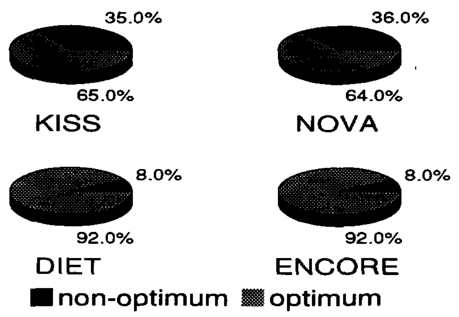


Fig. 2. Comparison of several encoding programs.

and are based on different theoretical foundations. The time complexities of the encoding algorithms used in these programs are *at best* quadratic in the size of the problem.

The second group of experiments gave very interesting results. Here we solve the partial constrained encoding problem: Given a bound on the encoding length, maximize the number of satisfied dichotomy constraints. The lengths chosen are the minimum ones needed to distinguish all the states. ENCORE produces better overall results than NOVA (cf. Table VII). For most of the FSM's, especially the *large* ones, the final PLA implementations occupy less area than those given by NOVA. Note that ENCORE aims at maximizing the number of satisfied dichotomy constraints, where NOVA aims at maximizing the number of satisfied group constraints.

VIII. Conclusions

There are two major results in this paper. First, we have developed an effective and efficient method for dichotomy-based constrained encoding—a problem fundamental to the synthesis of combinational and sequential logic circuits. Our method successfully combines the best features of the previous methods, and provides a unified solution to both complete and partial constrained encoding. In addition, our framework of constrained encoding can handle dominance constraints and disjunctive constraints arising from the output encoding problem.

Experiments with a number of applications indicate that our algorithm, as implemented in ENCORE, generates better results than the existing programs developed specifically in each application field. Since ENCORE is orders of magnitude faster, it is a promising alternative to existing techniques for solving large-size VLSI-CAD problems. We note that the lack of efficient methods for finding state assignments in asynchronous sequential synthesis has once been considered a major obstacle to the use of the asynchronous design methodology [3].

Second, it is demonstrated for the first time that synthesis results obtained using dichotomy constraints are comparable with the conventional group constraints in terms of PLA area used. We note that, while the reason for maximizing the number of satisfied group constraints is intuitively clear, the reason why maximizing the number of satisfied dichotomy constraints still yields the same result is not obvious. A theoretical analysis is needed as to improve our understanding of this aspect of sequential logic synthesis.

In addition, our framework of constrained encoding, which includes unary constraints as a special case of dichotomy constraints, allows us to formulate network partitioning [7], and via minimization [14], as two special cases (See [15]). Applications of the local search heuristic to via minimization is described in [14]. A further abstraction of constrained encoding is the signed hypergraph model described in [16]. It permits us to formulate these problems in a convenient graph-theoretic framework [16].

Appendix A
Formal Description of the Encoding Algorithm

The algorithm is presented in a top-down manner. The pseudo-code for the entire encoding algorithm, called GREEDY_ENCODING, is given below. The algorithm starts with an empty matrix **A**, and then invokes the procedure GENERATE_BIT to find a bit assignment α that satisfies as many constraints in C as possible. The process

TABLE VI
ENCODING WITH ALL THE DICHOTOMY CONSTRAINTS SATISFIED

	The Encoding Length							CPU-time(s)*		
FSM	#bits	#cbits	KISS	NOVA	DIET	ENCORE^{nb}	ENCORE^{b}	NOVA	DIET	ENCORE^{b}
dk15	2	4	4	4	4	4	4	3.2	0.1	0.00**
lion	2	—	—	—	2	2	2	—	0.1	0.00
mc	2	—	—	—	—	2	2	—	0.1	0.00
tav	2	2	—	—	2	2	2	—	0.1	0.01
train4	2	2	—	—	2	2	2	—	0.1	0.00
s8	3	3	—	—	3	3	3	—	0.1	0.00
bbtas	3	3	3	—	—	3	3	0.0	0.1	0.00
beecount	3	4	—	4	—	4	4	0.1	0.2	0.01
dk14	3	4	4	5	4	5	4	1.9	0.4	0.05
dk27	3	3	—	3	—	3	3	0.28	0.3	0.01
dk17	3	4	4	4	4	4	4	0.5	0.6	0.01
ex6	4	4	5	5	4	5	4	0.9	0.3	0.01
shiftreg	3	3	—	3	3	3	3	0.35	0.2	0.01
ex5	4	5	5	6	5	5	5	3.95	0.6	0.01
lion9	4	4	4	—	4	4	4	—	0.5	0.02
bbara	5	5	—	5	—	5	5	120.6	0.5	0.01
ex3	4	5	5	6	7	6	6	0.53	1.2	0.03
ex7	4	—	—	—	6	6	6	—	0.8	0.02
opus	4	4	—	—	—	4	4	—	0.5	0.00
train11	4	5	6	5	5	5	5	3.46	1.7	0.00
modulo12	4	4	—	—	—	4	4	—	1.4	0.00
ex4	4	—	—	—	4	4	4	—	2.5	0.02
dk512	4	5	6	6	5	5	5	35	4.0	0.03
mark1	5	—	5	—	4	4	4	29	2.5	0.01
bbsse	6	6	6	6	6	6	6	0.3	2.1	0.04
cse	4	5	6	5	5	6	5	0.2	3.4	0.03
kirkman	4	—	—	—	6	6	6	—	6.7	0.10
sse	4	—	—	—	6	6	6	—	2.1	0.02
ex2	5	6	6	6	6	7	6	2.46	8.6	0.05
keyb	5	7	8	7	8	9	8	28	7.7	0.10
ex1	5	7	7	7	7	7	7	35	11.2	0.12
s1	5	5	5	5	5	5	5	2.33	12.1	0.01
s1a	5	5	5	—	5	5	5	—	8.7	0.01
donfile	5	≤11	12	15	7	9	6	555	26.6	0.01
dk16	5	7	10	10	8	8	8	311	87.6	0.09
styr	5	6	6	9	6	8	6	52	26.4	0.15
sand	5	6	6	6	6	6	6	39.71	62.1	0.03
tbk	5	—	—	?	*f*	23	23	1301	*f*	0.14
planet	6	6	—	6	*f*	7	7	37.6	*f*	1.07
scf	7	≤8	8	—	*f*	8	8	—	*f*	0.53

*VAX-11/8650
**less than 0.01
— not applicable
? solution not found within the used CPU time
f failed
b with balance requirement
nb without balance requirement

is repeated until all the constraints are satisfied. A solution **A** is the concatenation of all the bit assignments found.

```
GREEDY_ENCODING($S$, $C$)
1   A←[ ]
2   while $C \neq$ NIL
3     do $\alpha \leftarrow$ GENERATE_BIT($S$, $C$)
4        $A \leftarrow [A; \alpha]$
5        $C \leftarrow C - \{$constraints satisfied by $\alpha\}$
6   return A
```

GENERATE_BIT: Lines 1 and 2 initialize the seed α. Line 3 invokes the procedure COMPUTE_GAIN to calculate the distances and to construct $\mathcal{B}$ under α. The return value of COMPUTE_GAIN is the number of unsatisfied constraints. In line 4, the component of highest gain is returned with the aid of the operation MAX on $\mathcal{B}$. If the gain value is positive, Lines 5–9 perform one move, that is, lock component i, remove the gain of component i from $\mathcal{B}$, invoke the procedure UPDATE_GAIN to update $\mathcal{B}$and the related distances due to the change of α_i, change α_i, and finally calculate the number u of unsatisfied constraints. Lines 5–9 repeat until either there is no positive-gain component, or all components are locked, i.e., $\mathcal{B}$ is empty.

COMPUTE_GAIN: Line 1 initializes the number u of unsatisfied constraints to n. Lines 2 and 3 initialize the temporary array γ to zero. Lines 4 to 23 form the main body for calculating the distances u and the gain vector

TABLE VII
COMPARISONS OF NOVA/ih, NOVA/ig AND ENCORE

	1-hot		NOVA/ih		NOVA/ig		ENCORE		random		
FSM	#cubes	#bits	#cubes	area	#cubes	area	#cubes	area	#cubes[1]	area[1]	area[2]
dk15	17	2	19	323	18	306	18	306	20	340	368
lion	8	2	6	66	6	66	7	77	7	77	96
mc	10	2	9	153	9	153	8	136	9	153	157
tav	12	2	11	198	11	198	11	198	11	198	198
train4	7	2	6	66	6	66	6	66	7	77	80
s8	14	3	10	180	10	180	9	162	9	162	198
bbtas	16	3	9	135	12	180	10	150	12	180	206
beecount	12	3	13	247	12	228	10	190	14	266	299
dk14	25	3	29	580	27	540	27	540	33	660	743
dk27	10	3	9	117	8	104	8	104	9	117	141
dk17	20	3	19	304	19	304	17	272	19	304	352
ex6	23	3	25	675	25	675	25	675	30	810	842
shiftreg	9	4	4	48	8	96	6	72	11	132	132
ex5	19	4	14	252	18	324	16	288	18	324	352
lion9	10	4	8	136	9	153	8	136	11	187	250
bbara	34	4	25	550	25	550	25	550	28	616	662
ex3	21	4	18	324	18	324	17	306	19	342	405
ex7	20	4	17	306	17	306	18	324	17	306	387
opus	19	4	16	448	16	448	18	504	20	560	581
train11	11	4	9	153	12	204	10	170	12	204	243
modulo12	24	4	12	180	12	180	14	210	12	180	195
ex4	21	4	19	627	19	627	21	693	19	627	683
dk512	21	4	18	306	19	323	19	323	22	374	419
mark1	19	4	21	798	19	722	18	684	19	722	786
bbsse	30	4	30	990	30	990	30	990	32	1056	1173
cse	57	4	46	1518	45	1485	45	1485	51	1683	2083
kirkman	61	4	79	3318	77	3234	61	2745	87	3654	4641
sse	30	4	30	990	30	990	30	990	32	1056	1176
ex2	38	5	29	609	36	756	32	672	38	798	905
keyb	77	5	48	1488	55	1705	51	1581	58	1798	3154
ex1	44	5	48	2496	51	2652	45	2475	60	3120	3281
s1	92	5	80	2960	87	3219	86	3182	96	3553	3703
s1a	92	5	76	2812	80	2960	73	2701	84	3108	3468
donfile	24	5	35	700	48	960	18	360	60	1200	1382
dk16	55	5	59	1298	72	1584	58	1276	87	1914	1981
styr	111	5	94	4042	103	4429	93	3999	126	5418	5691
sand	114	5	101	4646	102	4692	100	4600	93	4278	4972
tbk	173	5	154	4620	176	5280	128	3840	183	5490	6090
planet	92	6	91	4641	89	4539	90	4590	96	4896	5260
scf	151	7	148	19388	146	19126	140	18340	152	19912	21294
TOTAL				63688		65858		60979		70852	79029
%				90		93		86		100	112

[1]best random solution
[2]average of random solutions
#bits: code-length
#cubes: number of product-terms after ESPRESSO logic minimization
area: (2*(#inputs + #bits) + #bits + #outputs)*#cubes

γ. Lines 24 and 25 build $\mathcal{B}$ according to the calculated gain vector.

```
GENERATE_BIT($S$, $C$)
1  for $i \leftarrow 1$ to $m$
2  do $\alpha_i \leftarrow 1$
3  $u \leftarrow$ COMPUTE_GAIN($S$, $C$, $\alpha$, $\mathcal{B}$)
4  while (gain of MAX ($\mathcal{B}$) $> 0$ and $u > 0$)
5     do lock $\alpha_i$
6        DELETE ($\mathcal{B}$, $i$)
7        UPDATE_GAIN($S$, $C$, $i$, $\alpha$, $\mathcal{B}$)
8        $\alpha_i \leftarrow -\alpha_i$
9        $u \leftarrow u -$ MAXGAIN
10 return $\alpha$
```

```
COMPUTE_GAIN($S$, $C$, $\alpha$, $\mathcal{B}$)
1  $u \leftarrow n$
2  for $i \leftarrow 1$ to $m$
3     do $\gamma_i \leftarrow 0$
4  for each $c_j \in C$
5     do $d_j^+ \leftarrow d_j^- \leftarrow 0$
6        for each $s_i$ contained in $c_j$
7           do $\delta_{ij} \leftarrow \alpha_i c_{ij}$
8              if $\delta_{ij} = -1$
9                 then $d_j^+ \leftarrow d_j^+ + 1$
10                else $d_j^- \leftarrow d_j^- + 1$
11    $d_j \leftarrow \min\{d_j^+, d_j^-\}$
12    if $d_j = 0$
13       then $u \leftarrow u - 1$
```

```
14          for each $s_i$ in $c_j$
15              do $\gamma_i \leftarrow \gamma_i - 1$
16      if $d_j = 1$
17          then if $|C_i| = 2$
18              then for each $s_i$ in $c_j$
19                      do $\gamma_i \leftarrow \gamma_i + 1$
20              else for each $s_i$ in $c_j$
21                      do if $s_i$ is $\alpha_i$-sensitive
22                          then $\gamma_i \leftarrow \gamma_i + 1$
23                              break
24    for i $\leftarrow$ 1 to $m$
25      do INSERT($\mathcal{B}$. $i$. $\gamma_i$)
26    return $u$
```

MODIFY_GAIN: Given a constraint c_j, the distance d_j, and a parameter f, the procedure MODIFY_GAIN(f, c_j, d_j, $\mathcal{B}$) works as follows. If $f = 1$, the constraint's contribution is added to $\mathcal{B}$; if $f = -1$, the contribution is removed from $\mathcal{B}$.

```
MODIFY_GAIN($f$, $c_j$, $d_j$, $\mathcal{B}$)
    if $d_j = 0$
      then for each $s_i$ in $c_j$
              do UPDATE($\mathcal{B}$, $i$, $f$: $-f$)
      else if $d_j = 1$
              then if $|Ci| = 2$
                      then for each unlocked $s_i$ in $c_j$
                              do UPDATE($\mathcal{B}$, $i$, $f$)
                      else for each unlocked $s_i$ in $c_j$
                              do if $s_i$ is $\alpha_i$-sensitive
                                  then UPDATE ($\mathcal{B}$, $i$, $f$)
                                      break
```

```
UPDATE_GAIN($S$, $C$, $i$, $\alpha$, $\mathcal{B}$)
  for each $c_j$ in $C_i$
    do $\delta_{ij} \leftarrow \alpha_i c_{ij}$
      if ($d$ == 0 or 1) MODIFY_GAIN($-1$, $c_j$, $d_j$, $\mathcal{B}$)
      $d_j^+ \leftarrow d_j^+ + \delta_{ij}$
      $d_j^- \leftarrow d_j^- - \delta_{ij}$
      if ($d$ == 0 or 1) MODIFY_GAIN(1, $c_j$, $d_j$, $\mathcal{B}$)
```

Appendix B
Proof of Proposition 6

Initialization in Lines 1 and 2 takes m time. By Proposition 2, Line 3 takes $O(p)$ time. By Proposition 5, Line 4 takes $O(p)$ time. In the worst case, Lines 5–9 are repeated m times. Each time the procedure UPDATE is invoked, it takes $O(|C_i|)$ operations to do Lines 1, 2, 4, 5. According to Proposition 3, the number of calls to MODIFY_GAIN for constraint c_j during the entire GENERATE_BIT is a constant, say t_1. Each call to MODIFY_GAIN for constraint c_j takes $|c_j|$ operations. Therefore the number of operations required by Line 7 in GENERATE_BIT is $\Sigma_{i=1}^{m} O(|C_i|) + \Sigma_{j=1}^{n} (t_1|c_j|) = O(p) + t_1 p = O(p)$. Hence the time complexity of algorithm GENERATE_BIT is $O(p)$.

Acknowledgment

The authors would like to thank Professor Maciej Ciesielski of the University of Massachusetts for providing the program DIET. The authors wish to express their thanks to the three expert reviewers for their comments.

References

[1] J. I. Acha and J. Calvo, "On the implementation of sequential circuits with PLA modules," *Inst. Elect. Eng. Proc.*, vol. 132, pt. E., no. 5, pp. 246–250, Sept. 1985.

[2] R. K. Brayton, G. D. Hatchtel, C. T. McMullen and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis. NEws York: Kluwer Academic Publishers, 1984.*

[3] T. A. Chu, "Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications," Ph.D. dissertation, MIT, June 1987.

[4] S. Devadas and A. R. Newton, "Exact algorithms for output encoding, state assignment, and four-level Boolean minimization," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 13–27, Jan. 1991.

[5] S. Devadas, A. R. Wang, A. R. Newton, and A. Sangiovanni-Vincentelli, "Boolean decomposition of programmable logic arrays," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1988, pp. 2.5.1–2.5.5.

[6] T. A. Dolotta and E. J. McCluskey, "The coding of internal states of sequential machines," *IEEE Trans. Elect. Comput.*, vol. EC-13, pp. 549–562, Oct. 1964.

[7] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. of ACM/IEEE Design Automation Conf.*, 1982, pp. 175–180.

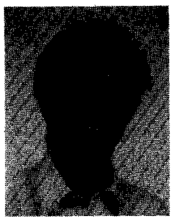
[8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness. San Franciscso, CA: Freeman, 1979.*

[9] G. D. Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimal state assignment for finite state machines," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, pp. 269–285, July 1985.

[10] G. D. Micheli, "Symbolic design of combinational and sequential logic circuits implemented by two-level logic macros," *IEEE Trans. Computer-Aided Design*, vol. CAD-5, pp. 597–616, Oct. 1986.

[11] R. L. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 727–750, Sept. 1987.

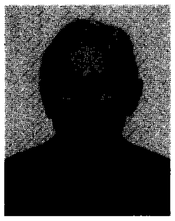
[12] A. Saldanha, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "A framework for satisfying input and output encoding constraints," in *Proc. ACM/IEEE Design Automation Conf.*, 1991, pp. 170-175.
[13] G. Saucier, C. Duff and F. Poirot, "State assignement using a new embedding method based on an intersecting cube theory," in *Proc. of ACM/IEEE Design Automation Conf.*, Las Vegas, June 1989, pp. 321-326.
[14] C.-J. Shi, "Constrained via minimization and signed hypergraph partitioning," in *Algorithmic Aspects of VLSI Layouts*, eds. D. T. Lee and M. Sarrafzadeh. River Edge, NJ: World Scientific, 1992.
[15] C.-J. Shi and J. A. Brzozowski, "An Efficient Algorithm for Constrained Encoding and Its Applications," Tech. Rep., No. CS-92-20, Dept. Comp. Sci., Univ. of Waterloo, Apr. 1992.
[16] C.-J. Shi and J. A. Brzozowski, "Graph-Theoretic Structures of Logic Encoding and Layer Assignment in VLSI Design," Tech. Rep., Dept. Comp. Sci., Univ. of Waterloo, in preparation.
[17] C.-J. Tan, "State assignment for asynchronous sequential machines," *IEEE Trans. Comp.*, vol. C-20, pp. 382-391, Apr. 1971.
[18] J. H. Tracey, "Internal state assignment for asynchronous sequential machines," *IEEE Trans. Electron. Comput.*, pp. 551-560, Aug. 1966.
[19] S. H. Unger, "A row assignment for delay-free realizations of flow tables without essential hazards," *IEEE Trans. Electron. Comput.*, vol. C-17, pp. 145-158, Feb. 1968.
[20] S. H. Unger, *Asynchronous Sequential Switching Circuits.* New York: Wiley, 1969.
[21] T. Villa and A. Sangiovanni-Vincentelli, "NOVA: State assignment of finite state machines for optimal two-level logic implementation," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 905-924, Sept. 1990.
[22] D. Varma and E. A. Trachtenberg, "A fast algorithm for the optimal state assignment of large finite state machines," in *Proc. of International Conf. on CAD*, 1988, pp. 152-155.
[23] S. Yang and M. J. Ciesielski, "Optimum and suboptimum algorithms for input encoding and its relationship to logic minimization," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 4-12, Jan. 1991.



C.-J. Richard Shi (S'91) received the B.Sc. and M.Sc. degrees in electrical engineering from Fudan University, Shanghai, China, in 1985 and 1987, respectively, and the M.A.Sc. degree in electrical engineering from the University of Waterloo, Waterloo, Canada, in 1991. Presently, he is completing his Ph.D. degree in Computer Science at the University of Waterloo.

Since 1983, he has been working on various aspects of VLSI-CAD, including circuit simulation, timing analysis, schematic capture, via minimization, logic synthesis, and sequential circuit testing. He has authored and coauthored more than twenty technical papers, and eight public domain or commerical CAD tools. His M.Sc. thesis research on timing verification was published in ICCAD'87 and ISCA'S88. From 1988 to 1989, he was a faculty member at Fudan University, and was Project Leader of the Timing Analysis Group at the Beijing IC Design Center for PANDA, a key national project in China. In 1990, he was consulting on CADENCE/EDGE tools and CMC chip fabrication for the University of Waterloo. In 1991, he was a research associate with the Department of Computer Science at the University of Waterloo. His research interests are in the area of VLSI-CAD, with current emphasis on synthesis and testing of sequential circuits.

Mr. Shi received the T.D. Lee Physics Award for excellence in graduate research from Fudan in 1987, and a best paper award from the Shanghai Science and Technology Association in 1988.



Janusz A. (John) Brzozowski (M'87) received the B.A.Sc. and M.A. Sc. degrees in electrical engineering from the University of Toronto in 1957 and 1959, respectively, and the M.A. and Ph.D. degrees in electrical engineering from Princeton University in 1962.

He was Assistant Professor from 1962 to 1965 and Associate Professor from 1965 to 1967 in the Department of Electrical Engineering, University of Ottawa. Since 1967 he has been a Professor in the Department of Computer Science, University of Waterloo. In the periods 1978-1983 and 1987-1989 he was chair of that department. He has had visiting appointments at the University of California, Berkeley (1965-1966), University of Paris (1974-1975), University of São Paulo (1983), Kyoto University (1984), and Eindhoven University (1989-1990). He has published many papers in the areas of algebraic theory of regular languages, finite automata, asynchronous circuits, and testing. He is also a co-author of *Digital Networks* (Englewood Cliffs, NJ: Prentice-Hall, 1976). His present research interests include asynchronous circuits, delay-insensitive design, VLSI models, testing, automata and formal languages.

Dr. Brzozowski is a member of ACM, IEEE, EATCS, and the Association of Professional Engineers of Ontario.