A Fast and Robust Exact Algorithm for Face Embedding

Evguenii I. Goldberg[†,‡] Tiziano Villa[§] Robert K. Brayton[‡]

Alberto L. Sangiovanni-V incentelli[‡]

[‡] Department of EECS
University of California at
Berkeley,Berkeley, CA 94720

[†] Academy of Sciences of
Belarus, Minsk

[§] PARADES,
Via di S.Pantaleo,
66, 00186 Roma

Abstract

We present a new matrix formulation of the face hypercube embedding problem that motivates the design of an efficient search strategy to find an encoding that satisfies all faces of minimum length. Increasing dimensions of the Boolean space are explored; for a given dimension constraints are satisfied one at a time. The following features help to reduce the nodes of the solution space that must be explored: candidate cubes instead of candidate codes are generated, cubes yielding symmetric solutions are not generated, a smaller sufficient set of solutions (producing basic sections) is explored, necessary conditions help discard unsuitable candidate cubes, early detection that a partial solution cannot be extended to be a global solution prunes infeasible portions of the search tree.

We have implemented a prototype package MINSK based on the previous ideas and run experiments to evaluate it. The experiments show that MINSK is faster and solves more problems than any available algorithm. Moreover, MINSK is a robust algorithm, while most of the proposed alternatives are not. Besides most problems of the complete MCNC benchmark suite, other solved examples include an important set of decoder PLAs coming from the design of microprocessor instruction sets.

1 Introduction

Consider a set of symbols S and an **encoding** function $e : S \rightarrow B^k$, for a given k, that assigns to each symbol $s \in S$ a code $e(s)$, i.e., a binary vector of length k. Usually the only requirement is that e is injective, i.e., that different symbols are mapped to different binary vectors. In various applications it is important to satisfy other encoding constraints, in order to obtain a code that is correct or desirable to meet a certain objective. Either the encoding length k is part of the problem instance or it is an unknown to be found (usually minimized) by the procedure that satisfies the given encoding constraints [15].

Given a set of symbols S, a **face constraint** c_f is a subset $S' \subseteq S$ specifying that the symbols in S' are to be assigned to one *face* (or subcube) of a binary k-dimensional cube, without any other symbol sharing the same face. Face constraints are generated by multiple-valued (input) literals in two-level and multi-level multi-valued minimization [15]. As an example, given symbols a, b, c, d, e, an input constraint involving symbols a, b, c is denoted by (a, b, c). An encoding satisfying (a, b, c) is given by $a = 111$, $b = 011$, $c = 001$, $d = 000$ and $e = 100$ and the face spanned by (a, b, c) is $- - 1$. Notice that the vertex 101 is not and should not be assigned to any other symbol.

Given a set of face constraints $\mathcal{C}_f$, it is always possible to find an encoding that satisfies it, as long as one is free to choose a suitable code length. It is a well-known fact that for $k = \mid S \mid$ any set $\mathcal{C}_f$ is satisfied by choosing as e the *1-hot* encoding function (which assigns to a state s_i the binary vector that is always 0 except for a position to 1, the latter denoting state s_i). It is an important combinatorial optimization problem, sometimes called [16] **face hypercube embedding**, to find the minimum k and a related $e : S \rightarrow B^k$ such that $\mathcal{C}_f$ is satisfied. The decision version of this problem is NP-complete [11].

An exact solution based on a branch-and-bound strategy to search the partially ordered set of faces of hypercubes was described first in [16], but it is not computationally practical. An exact solution by reduction to the problem of satisfaction of encoding dichotomies [1] was proposed in [17]. It uses a reduction by J. Tracey [14] of the exact satisfaction of encoding dichotomies to a unate covering problem. This approach was made more efficient in [11], by improving the step of generating maximal compatibles of encoding dichotomies. Recently the problem of satisfaction of encoding dichotomies has been revisited in [3], adapting techniques to find primes and solving unate covering with binary decision diagrams that have been so successful in two level logic minimization [2]. From the experimental point-of-view none of the previous algorithms has performed up to expectations, being unable to solve exactly various instances of moderate size and practical interest. Moreover, algorithms reducing encoding dichotomies to unate covering have a dismal behavior when the problem instance consists mostly of uniqueness encoding dichotomies (i.e., encoding dichotomies with only one state in each block), because they generate most of the encoding columns, which are 2^k for $k = \mid S \mid$.

Heuristic solutions to the face embedding problem have been reported in many papers [10, 4, 12, 5, 17, 13]. A heuristic solution satisfies all face constraints, but does not guarantee that the code-length is minimum. A related problem, that is not of interest in this paper, is the one of fixing the code-length and maximizing a gain function of the constraints that can be satisfied in the given code-length. We refer to [15] for background material on satisfaction of encoding constraints and their sources in logic synthesis.

In this paper we present a new matrix formulation of the face hypercube embedding problem that inspires the design of an efficient exact search strategy. This algorithm satisfies the constraints one by one by assigning to them intersecting cubes in the encoding Boolean space. The problem of finding a set of cubes with a minimum number of coordinates satisfying a given intersection matrix was first formulated in [18] without any relation to encoding problems. No algorithm to solve the problem was described. The relation between the face embedding problem and the construction of intersecting cubes was employed in an heuristic algorithm described in [12, 5]. The first formulation of a simple criterion of when a set of cubes satisfies a set of constraints was given in [6]. We use some theoretical notions, e.g., basic sections, introduced

[1] An encoding dichotomy on S is a bipartition (S_1, S_2) such that $S_1 \cup S_2 \subseteq S$.

0-89791-993-9/97 $10.00

first in [7, 8]. The following features speed up the search of our algorithm: candidate cubes instead of candidate codes are generated, symmetric cubes are not generated, a smaller sufficient set of solutions (producing basic sections) is explored, necessary conditions help discard unsuitable candidate cubes, early detection that a partial solution cannot be extended to be a global solution prunes infeasible portions of the search tree. The experiments with a prototype implementation in a package called MINSK show that our algorithm is faster, solves more problems than any available alternative and is robust. All problems of the MCNC benchmark suite were solved successfully, except four of them unsolved or untried by any other tool. Other collections of examples were solved or reported for the first time, including an important set of decoder PLAs coming from the design of microprocessor instruction sets.

In Section 2 we present a theoretical formulation based on matrix notation. How to avoid the generation of symmetrical solutions is explained in Section 3. In Section 4 we describe a new algorithm to satisfy face constraints. Experimental results are provided in Section 5. Section 6 concludes the paper with remarks on what has been achieved and future work.

2 Matrix Formulation of the Face Embedding Problem

Given a matrix M, denote by $Row(M)$ its rows and $Col(M)$ its columns. $M_{i,}$ denotes the i-th row of M and $M_{,j}$ denotes the j-th column of M. The **multiplicity** of a column $C_{,j}$ of M, $mult(j)$ is the number of times that $C_{,j}$ occurs in M. We use the term vector to indicate a one dimensional matrix, when there is no need to specify whether it is regarded as a row or a column. Vectors are called **binary** or **two-valued** if their entries are 0 or 1 and **3-valued** if their entries are 0 or 1 or $-$. A **singleton** vector has a unique 1.

Given two 2-valued vectors v_1 and v_2 of the same length, their **disjunction** $v_1 \cup v_2$ is the vector v whose i-th entry is the disjunction of the i-th entries of v_1 and v_2. Similar definition holds for the **conjunction** of v_1 and v_2. A vector v_1 **covers** a vector v_2 if, whenever the i-th entry of v_2 is 1, the i-th entry of v_1 is 1. A vector v_1 **intersects** a vector v_2 if for at least an index i, the i-th entry of v_1 and v_2 is 1.

2.1 Constraint and Solution Matrices

Given a set of symbols S and a set of face constraints $\mathcal{C}_f$ on S, the **constraint matrix** is a matrix with as many rows as constraints and columns as symbols. Entry (i, j) is 1 iff the i-th constraint contains symbol j, otherwise it is 0. For don't care face constraints, the don't care states have a $-$ in the corresponding position of the constraint matrix.

Consider the set of constraints $\mathcal{C}_f = \{(s_3 s_4 s_6 s_9), (s_3 s_5), (s_1 s_4 s_7), (s_2 s_3 s_6), (s_7 s_8), (s_{11} s_{12})\}$. Then the related constraint matrix is:

Example 2.1

$$C = \begin{array}{|cccccccccccc|} \hline s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 & s_9 & s_{10} & s_{11} & s_{12} \\ \hline 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ \hline \end{array}$$

In the sequel we will refer usually to a set of face constraints $\mathcal{C}_f$ by its encoding matrix C and we will not distinguish the two. Notice that there is no need to add singleton constraints, because we guarantee that different codes are assigned to different states, including the states whose columns in $\mathcal{C}_f$ are equal.

Given an encoding e that satisfies a constraint matrix C, e defines a face for each constraint of C, i.e., the minimum subcube that contains the codes of the states in the constraint.

For a given constraint matrix C and integer n, consider a **face matrix** S with $Row(C)$ rows (**faces** or **cubes**) and n columns (**sections**), whose entries may be 0 or 1 or $-$. Each row may be regarded as a subcube in the n-dimensional Boolean space. If there exists an encoding e such that, for each $i \in Row(S)$, the i-th row of S is the face that e defines for the i-constraint of C, then we say that S is a **solution face matrix** of C or that S **satisfies** C and that the i-th row of S is a **solution cube** of the i-th constraint.

One verifies that S is a solution face matrix of C, by constructing an **intersection matrix** T_S whose rows are the cubes of S and whose columns are the minterms of B^n, where entry (i, j) is 1 iff minterm j is in cube i. Then S satisfies C if for any column $C_{,j}$, the matrix T_S contains no less than $mult(C_{,j})$ columns equal to $C_{,j}$. In other words, we require that each minterm (code of a state) belongs only to those faces to which it is restricted by the constraints; moreover, if there are equal columns in the constraint matrix, for each of them there must be a different minterm. In this way, there is at least one injective function $f_{C \to T_S}$ that associates to each column of C one column of T_S.

Given a matrix S satisfying C, an encoding e_S that satisfies C can be extracted with the following rule: select an injective function $f_{C \to T_S}$, whose existence is guaranteed because S satisfies C, then encode state i (i.e., column i of C) with the minterm of the column $f_{C \to T_S}(i)$ in T_S. Such an encoding satisfies C because each code lies only in the faces corresponding to the constraints to which the state belongs.

Example 2.2 *Given the previous C and $n = 4$, consider*

$$S = \begin{bmatrix} - & 0 & 1 & - \\ 1 & 0 & - & 0 \\ 1 & - & - & 1 \\ - & - & 1 & 0 \\ - & 1 & 0 & 1 \\ 0 & - & 0 & 0 \end{bmatrix}$$

S satisfies C as it is shown by building the matrix

$$T_S = \begin{array}{|cccccccccccccccc|} \hline 1001 & 0110 & 1010 & 1011 & 1000 & 0010 & 1101 & 0101 & 0011 & 0100 & 0000 & 0111 & 0001 & 1100 & 1111 & 1110 \\ \hline 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

An encoding e_S that satisfies C can be extracted from S, with the following injection from columns of C to columns of T_S: $e(s_1) = 1001$, $e(s_2) = 0110$, $e(s_3) = 1010$, $e(s_4) = 1011$, $e(s_5) = 1000$, $e(s_6) = 0010$, $e(s_7) = 1101$, $e(s_8) = 0101$, $e(s_9) = 0011$, $e(s_{10}) = 0100$, $e(s_{11}) = 0000$, $e(s_{12}) = 0111$.

Notice that columns of T_S corresponding to the following groups of codes $\{0100, 0000\}$, $\{0111, 0001, 1100\}$, $\{1001, 1111\}$, $\{0110, 1110\}$ are equal, which gives freedom in selecting an encoding satisfying C. For example, if we permute codes 0100 and 0000 in the previous encoding e_S or assign to $e(s_{12})$ code 0001 instead of 0111 we still get an encoding satisfying C. So many different encodings satisfying C specify the same set of faces.

2.2 Basic Sections

Given a constraint matrix C and an encoding e, the set of the minimal cubes such that each of them contains the codes of the symbols in a corresponding constraint of C defines the rows of a face matrix S. If e satisfies C then S is a solution face matrix of C. The operation of a finding the minimal cube that contains the codes of the states that appear in a given constraint is captured

exactly by the notion of basic section that we are going to define next. Informally, given a constraint matrix C, the columns of a face matrix S such that there is an encoding e (that may or may not satisfy C) for which the rows of S are the minimal cubes containing the codes of the constraints of C are basic sections.

Consider a vector d (whose elements are 0 or 1) with $|Col(C)|$ entries. We can regard d as an **encoding column**, i.e., an assignment of 0 or 1 to each symbol. An encoding function $e : S \rightarrow B^k$ defines a set of k encoding columns $e_1, \cdots, e_k$ (i.e., the columns of e), where the i-th entry of e_j is 1 (is 0) if and only if the j-th coordinate of $e(s_i)$ is 1 (is 0).

Let us compare the set of columns that have a 1 in $C_{i.}$ (i-th row of C) with the set of columns that have a 1 in d. There are the following cases:

1. d covers $C_{i.}$, i.e., all columns that have a 1 in $C_{i.}$ have a 1 in d. Say that the comparison returns a 1.
2. d does not intersect $C_{i.}$, i.e., no column that has a 1 in $C_{i.}$ has a 1 in d. Say that the comparison returns a 0.
3. d intersects but does not cover $C_{i.}$, i.e., a proper subset of the columns that have a 1 in $C_{i.}$ have a 1 in d. Say that the comparison returns a $-$.

So given a d, let us denote by $bs(d)$ a column vector with $|Row(C)|$ entries of value 1, 0, or -, where the i-th entry is 1, 0 or $-$, according to whether the previous comparison of d and $C_{i.}$ returns a 1, 0 or $-$.

Definition 2.1 *A 3-valued column B is called a* **basic section** *for C if there is a vector d such that $B = bs(d)$.*

Example 2.3 *Consider*

$$C = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

The encoding $e(s_1) = 000$, $e(s_2) = 100$, $e(s_3) = 110$, $e(s_4) = 111$, $e(s_5) = 101$ satisfies C. The minimal cubes containing the codes assigned by e to the symbols in each constraint of C define a face matrix S:

$$S = \begin{bmatrix} - & 0 & 0 \\ 1 & - & 0 \\ 1 & 1 & - \\ 1 & - & 1 \end{bmatrix}$$

which satisfies C as seen by building the intersection matrix

$$T_S = \left[\begin{array}{c|cccccccc} & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\ \hline -00 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1-0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 11- & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1-1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{array}\right]$$

The encoding columns of the encoding are $e_1 = 01111$, $e_2 = 00110$, $e_3 = 00011$, and they yield the basic sections $bs(e_1) = -111$, $bs(e_2) = 0-1-$, $bs(e_3) = 00-1$. The rows of the matrix of the basic sections are the minimal cubes spanned by the codes of the states in the constraints of C.

Theorem 2.1 *Given a constraint matrix C and an encoding e with encoding columns $e_1, \ldots, e_k$, the basic sections $bs(e_1), \ldots, bs(e_k)$ define the set of minimal cubes such that each of them contains the codes of the symbols in a corresponding constraint of C (even if e does not satisfy C). Moreover, if e satisfies C the basic sections $bs(e_1), \ldots, bs(e_k)$ define a face matrix that satisfies C.*

It is worthwhile to clarify that a matrix S that satisfies a constraint matrix does not consist necessarily (only) of basic sections. A trivial case comes from "redundant" solutions, obtained by adding to a solution matrix S an arbitrary column (so not necessarily a basic section). A more interesting case comes from a solution matrix S whose faces are not minimal subcubes yielded by a corresponding encoding e. However, the following theorem shows that it is sufficient to consider only basic sections to find a minimum solution to face hypercube embedding.

Theorem 2.2 *Given a solution face matrix S' of the constraint matrix C there is always a solution face matrix S of C with the same number of columns that consists only of basic sections.*

Proofs of the theorems are omitted for lack of space.

3 Characterization of Symmetric Solutions

A crucial feature of an efficient algorithm to solve face embedding constraints is the ability to avoid the consideration of symmetric solutions, i.e., solutions that differ only by permutations and inversions of variables of the encoding space. We will refer to permutations and inversions of variables as symmetric transformations or symmetries.

If a matrix S is a solution of C then any matrix S' obtained from S by permutations and (bit-wise) inversions of columns is a solution of C. So if for example, in matrix S there are no columns that are equal or equal after inversion then there exist $n!\ 2^n$ ($n!$ for permutations and 2^n for inversions) different solutions obtained by symmetries of S. For example for $n = 6$ ($n = 7$) $n!\ 2^n$ is equal to 46080 (645120). Generation of such solutions is useless because they all have the same cost as S. All solutions produced by symmetries of S form an equivalence class of which it suffices to consider a representative to solve the problem.

In Section 4 we will present a procedure *search_boolean_space* that finds a solution face matrix S in an n-dimensional boolean space, if such a solution exists. Let us write $S^{(i)}$ for a solution face matrix satisfying the matrix $C^{(i)}$, which stands for the matrix C restricted to the first i rows.

The procedure builds incrementally a matrix S by finding first a solution $S^{(1)}$ for the first constraint C_1, then augmenting it to a solution $S^{(2)}$ for constraints C_1,C_2 and so on, until all constraints are considered. More precisely, when handling the i-th constraint, one generates the set $Sat(C_i)$ of all cubes satisfying it and selects a cube $F \in Sat(C_i)$. Then one verifies whether $S^{(i)}$ formed by appending cube F to $S^{(i-1)}$ satisfies $C^{(i)}$. If not, another cube of $Sat(C_i)$ is tried and, if none works, one backtracks further to a different choice of a cube $F' \in Sat(C_{i-1})$ such that $C^{(i-1)}$ is satisfied by $S^{(i-2)}$ augmented by F'.

Now we show how to avoid the generation of symmetrical solutions. Let $S^{(i-1)}$ be a face matrix satisfying $C^{(i-1)}$. Let A be a set of column numbers and suppose that all columns of $S^{(i-1)}$ from A are equal. Let F',F'' be n-component cubes which differ only in the columns from A. Denote by $S'^{(i)}$ and $S''^{(i)}$ the solutions obtained by augmenting $S^{(i-1)}$ by cubes F' and F'' respectively. If there is a permutation of columns from A transforming F' to F'' then the same permutation transforms $S'^{(i)}$ into $S''^{(i)}$, i.e., solutions $S'^{(i)}$ and $S''^{(i)}$ are symmetrical. It is not hard to show that cube F' can be transformed into cube F'' by a permutation of columns from A if and only if both cubes have the same numbers of 0 and 1 entries in the columns from A.

Assume that the columns of $C^{(i-1)}$ from A are not only equal, but also they do not contain 0 and 1 entries. Suppose that there is a permutation and inversion of columns from A transforming F' into F''. Then the same permutation and inversion transform $S'^{(i)}$ into

$S''^{(i)}$. It is not hard to show that cube F' can be transformed to F'' by a permutation and inversion of columns from A if and only if both cubes have the same number of - entries in the columns from A.

Example 3.1 *Consider C given in Example 2.1 for $n = 4$. When generating the solutions of $C^{(1)}$, i.e., cubes satisfying C_1 we need to generate only cubes having different number of - entries since matrix $C^{(0)}$ is empty, which can be interpreted as the case of all columns of $C^{(0)}$ equal and not containing 0 and 1 entries. So the candidate cubes to satisfy the first constraint are: $----$ (4 dashes),* $1---$ *(3 dashes),* $11--$ *(2 dashes),* $111-$ *(1 dash),* 1111 *(0 dashes). None of them can be obtained by a symmetric transformation of another and any other cube of 4 components can be obtained from one of the cubes above by a permutation and inversion of components . Note that there is freedom in selecting cubes representing an equivalence class. So instead of $11--$ we could select any cube having two dashes, for example $0--1$.*

Suppose that we have already chosen cube $11--$ to solve $C^{(1)}$, i.e., to satisfy the first constraint. So $S^{(1)}$ consisting of cube $11--$ has two groups of equal columns: $\{1, 2\}$ and $\{3, 4\}$. Note that the columns of the second group do not contain 0 and 1 entries. Generating cubes satisfying C_2 we want to avoid considering symmetrical solutions $S^{(2)}$. To do so we must skip the generation of cubes that have the same number of 0 and 1 entries in the columns of the first group and the same number of - in the columns of the second group. There are six patterns of two components having different number of 0 and 1 entries : $--$ *(0 zeroes, 0 ones),* $1-$ *(0 zeroes, 1 ones),* $0-$ *(1 zeroes, 0 ones),* 11 *(0 zeroes, 2 ones),* 00 *(2 zeroes, 0 ones),* 01 *(1 zeroes, 1 ones). These patterns give 6 possible combinations of the values of the first and second components of candidate cubes. There are 3 patterns of two components having different number of dashes:* $--$ *(2 dashes) ,* $1-$ *(1 dash),* 11 *(0 dashes). These patterns give 3 possible combinations of the values of the third and fourth components of candidate cubes.*

All together we obtain $6 \times 3 = 18$ combinations: $----$, $---1-$, $---11$, $1---$, $1--1-$, $1--11$, $11--$, $111-$, 1111, $0---$, $0--1-$, $0--11$, $01--$, $011-$, 0111, $00--$, $001-$, 0011. *Augmenting $S^{(1)}$ by a cube from the previous set we obtain 18 different matrices none of which can be obtained by symmetrical transformation from another.*

4 An Exact Algorithm to Find a Minimum Solution

In Fig. 1 we present the flow of an algorithm *find_solution* that finds a minimum solution of a constraint matrix C. It starts with the minimum dimension (log of the number of constraints) and it increases it until a solution is found. It is guaranteed to terminate because every constraint matrix can be satisfied by an encoding of length k, if k is the number of symbols; more precisely by an 1-hot encoding. Usually a much shorter encoding length suffices.

4.1 The Search Strategy

The key feature of the proposed algorithm is that it searches sets of cubes, instead of sets of codes. Since a set of cubes may correspond to many sets of codes (see Example 2.1), the algorithm explores simultaneously many encodings. Once a satisfactory set of cubes is found, it is straighforward to extract from it a satisfying encoding.

For a given dimension, the search of a satisfying encoding is carried through by the routine *search_space*, that returns a solution face matrix S that satisfies C. Once S is known, it is easy, as shown in Example 2.2, to find an encoding of the symbols that satisfies C.

Before calling *search_space* the constraints are ordered as mentioned in Section 4.5 and then processed in that order. Each call of *search_space* processes a new constraint. *Seach_space* keeps a current partial solution $Curr\_Sol$ that satisfies all the constraints from the first to the last constraint that has been processed. It satisfies a constraint by generating a cube that encodes the constraint (a row of S). A constraint is satisfied if there is a cube such that, by adding it to the current solution, we satisfy the constraint matrix restricted to the constraints from the first to the one currently processed.

Once the current constraint has been satisfied the current solution is updated and *search_space* calls itself recursively with a new constraint. If the current solution cannot be extended to satisfy the current constraint, *search_space* backtracks and tries a different cube for the last constraint that was satisfied by $Curr\_Sol$ and it continues to backtrack until it finds a partial solution $Curr\_Sol$ which can be extended to satisfy the constraint currently processed. The procedure *found_solution* tests whether a face matrix is a solution of a set of constraints, by constructing the intersection matrix T_S as shown in Section 2.1.

The following enhancements reduce the nodes of the search tree that *search_space* has to explore to find a minimum solution:

1. Candidate cubes are generated by a procedure *generate_cand_cubes* that avoids the generation of symmetric solutions, based on the theory presented in Section 3.

2. The procedure *generate_cand_cubes* eliminates also the cubes that would yield a matrix S with sections which are not basic, as allowed by Theorem 2.2 and explained in Section 4.2.

3. Cubes that do not satisfy the necessary conditions of Section 4.3 to be valid extensions of the current solution are discarded by a procedure *discard_cand_cubes*.

4. When trying to extend the current solution, the procedure *unsat_constr* checks first whether any of the constraints not yet processed is unsatisfiable by an extension of the current solution; if so, *search_space* backtracks to modify the current solution. See Section 4.4 for more discussion.

4.2 Restriction to Basic Sections

In Section 2 we highlighted the fact that not all solution face matrices S consist entirely of basic sections, but we argued in Theorem 2.2 that basic sections are sufficient to find a minimum solution. Therefore when generating cubes that are candidate solutions of face constraints it is profitable to reject those that would produce an S with some sections which are not basic. The rejection is performed in the following way.

Suppose that the algorithm is processing constraint C_i. At this time a solution $S^{(i-1)}$ satisfying $C^{(i-1)}$ is known. Suppose that all 3-valued columns of $S^{(i-1)}$ are basic sections. Adding to $S^{(i-1)}$ a cube F we extend each column of $S^{(i-1)}$ by one entry. The j-th column of $S^{(i-1)}$ can be extended in three ways according to the j-th component of F being equal to 0,1 or -. E.g., suppose that if the j-th component of F is equal to 0 then the j-th column of $S^{(i)}$ is not a basic section. Then when constructing cubes satisfying C_i we need to avoid the generation of those that have 0 in the j-th component.

So to guarantee that all produced solutions $S^{(i)}$ consist only of basic sections we need to generate cubes which correspond to "correct" extensions of columns of $S^{(i-1)}$. According to Definition 2.1, a 3-valued column is a basic section if there is a boolean vector d such that $bs(d)$ is equal to the column. We use a branch-and-bound algorithm that, given a column, checks whether the column is a basic section. We do not report here the details of the algorithm.

```
find_solution(C) {
    /* order the constraints */
    C = sort_constraints(C)
    for (cube_size = lg⌈| C |⌉; TRUE ; cube_size++) {
    cur_constr = 1  /* cur_constr is the index of the current constraint to satisfy */
    Sol = search_space(C, cube_size, ∅, cur_constr, {(1, ..., cube_size))}
    if Sol = ∅
        continue
    else
        return Sol
    }
}

search_space(C, cube_size, Curr_Sol, cur_constr, Classes) {
    /* Curr_Sol satisfies all constraints */
    if cur_const >| C |
        return Curr_Sol
    /* early detection of unsatisfiable constraints given Curr_Sol */
    if unsat_constr(C, Curr_Sol, cur_constr)
        return ∅
    /* generate candidate cubes CCubes excluding symmetric solutions
    and enforcing that all solutions consist only of basic sections */
    CCubes = generate_cand_cubes(C, cube_size, cur_constr, Curr_sol, Classes)
    /* sort candidate cubes in order of increasing size */
    CCubes = sort_cand_cubes(CCubes)
    /* eliminate candidate cubes that cannot satisfy constraints */
    CCubes = discard_cand_cubes(C, cur_constr, Curr_sol, CCubes)
    /* find a cube extending Curr_Sol to satisfy also current constraint */
    for (cur_cube = 1; i <| CCubes |; cur_cube++) {
        New_Curr_Sol = Curr_Sol ∪ cur_cube
        /* test if New_Curr_Sol satisfies constraints from 1 to cur_constr */
        if not found_solution(C, cur_constr, New_Curr_Sol)
            continue  /* not a solution: try another cube */
        /* solution found: recompute equivalence relation on columns */
        /* try to extend current solution to satisfy also next constraint */
        Sol = search_space(C, cube_size, New_Curr_Sol, cur_constr + 1, New_Classes)
        if Sol ≠ ∅
            return Sol
    }
    return ∅  /* current solution cannot be extended to satisfy also current constraint */
}
```

Figure 1: Algorithm to find a minimum solution.

4.3 Removal of Unsuitable Cubes

Let $S^{(i-1)}$ be a partial solution and $Cand(C_i)$ be a set of candidate cubes for satisfying C_i that do not contain symmetric cubes nor cubes leading to sections that are not basic. Before checking if $S^{(i-1)}$ together with a cube $F \in Cand(C_i)$ is a solution of $C^{(i)}$, it is worthy to test whether F satisfies some necessary conditions. Precisely we discard a cube $F \in Cand(C_i)$ it at least one of three conditions holds:

1. The number of 1s in C_i is greater than 2^n where n is the number of $-$s in F.
2. There is a k such that cube F covers the cube specified by the k-th row of $S^{(i-1)}$, i.e., F satisfies C_k, but row C_i. does not cover (dominate) row C_k.. In this case there is a column $C_{.m}$ of C such that $C_{im} = 0$ and $C_{km} = 1$, that does not appear in the intersection matrix of the set of cubes obtained by adding cube F to $S^{(i-1)}$.
3. There is a k such that C_k. intersects C_i., but the number of 1s in their intersection is greater than 2^n where n is the number of $-$s in the cube obtained by the intersection of F and the cube specified by the k-th row of $S^{(i-1)}$.

4.4 Early Detection of Unsatisfied Constraints

Constraints are processed one by one in a predefined order. Suppose that on the path leading to the current node of the search tree we have already chosen 4 cubes satisfying the first 4 constraints and that now we are trying to satisfy the 5-th constraint. Suppose also that all constraints from the 5-th to the 19-th are satisfiable, but that the 20-th is unsatisfiable, given the current choice of the first 4 cubes. So checking the satisfiability of one constraint at a time, we would discover that the 20-th constraint is unsatisfiable only after having processed all constraints up to the 19-th one; then we would start backtracking to another cube satisfying the 19-th constraint and we would try again to satisfy the 20-th one, and so on for all

the cubes that satisfy the 19-th constraint. We would repeat this time-consuming process for all constraints from to the 19-th to the 5-th one, before discovering that we must modify the solution to the first 4 constraints, in order to extend it to a solution that satisfies the constraints up to the 20-th one.

To prevent such unrobust behaviour and lessen the dependency on how the constraints are sorted initially, we employ early detection of unsatisfied constraints. At each node of the search tree with i satisfied constraints, the algorithm checks first that any of the remaining unprocessed constraints is satisfiable, given the current choice of cubes which satisfy the first i constraints. Although this checks requires some extra calculations at each node of the search tree, it is fully justified by the drastic reduction of the search tree size.

4.5 Sorting of Constraints

Constraints are sorted with the goal to prune branches of the search tree at the earliest possible stages. We have two sorting criteria. The first one selects as next constraint the one that intersects the highest number of already selected constraints. Ties are broken selecting the constraint with the highest number of 1s. The second criterion selects according to the highest number of 1s and breaks ties with the highest number of intersected rows.

5 Results

We implemented the algorithm described in Section 4 in a prototype package in C called MINSK (Minimum INput Satisfaction Kernel) and we applied it to a set of benchmarks available in the literature. The benchmarks are partitioned into three sets: FSMs from the MCNC collection, reported in Table 2; FSMs collected from various other sources, reported in the upper part of Table 4; decode PLAs of the VLSI-BAM project, provided by Bruce Holmer [9], reported in the lower part of Table 4. In all cases, face constraints were generated with ESPRESSO [1]. In the tables we report: the name of the example, the number of symbols to encode ("#states"), the logarithm of the number of symbols ("min. len.") together with the minimum code length to satisfy all input constraints known so far ("best known"), and the minimum code length to satisfy all input constraints found by MINSK ("min. sol."). Besides, the tables show the number of calls of the routine *found_solution* ("#checks"), the number of recursive calls of the routine *search_space* ("#calls"), and the CPU time for a 300 Mhz DEC ALPHA workstation. We did not report data on examples where the constraints were few and MINSK found a solution in no time. We found an exact solution for all the examples, except the FSMs *tbk, s1488, s1494, s298* none of which has been solved before. For some examples, like *donfile, scf, dk16* exact solutions were never found automatically before; for others, like *ex2* an exact solution was found automatically by NOVA [16] with the option *-e ie*, but at the cost of an unreasonable CPU time (60172.6 s. on 60Mhz DEC RISC workstation) [2].

Up to now four exact algorithms have been tried to solve face hypercube embedding. The first is available as an option in NOVA *-e ie*, the second is based on a reduction to satisfaction of encoding dichotomies by means of unate covering [17, 11], the third is an implicit implementation with ZBDDs of the latter [3], and the last is a simplification of the third, where instead of prime dichotomies one uses all possible encoding dichotomies [3]. In Table 3 we compare the performance of MINSK with the last three previous algorithms, based on the data recently reported in [3]. We are aware that the experiments presented in [3] were run with a 75 Mhz SuperSparc workstation with 96 MB memory and a timeout of 2 hours. The purpose of the comparison is to evaluate the behaviors of the various algorithms, not to discuss specific running times. We included in Table 3 all the interesting examples, leaving out "easy" cases where all algorithms behaved similarly.

The experiments warrant the following practical conclusions:

- MINSK is a robust algorithm, that solves in no time problems with few constraints and requires more time when the set of constraints is larger and more difficult.
- MINSK is also superior in running times to the other programs in the more difficult cases, showing that the key ingredients of its search strategy, such as generating cubes and not codes, avoiding symmetric solutions and sections which are not basic, prune away large suboptimal portions of the search space. The exact option of NOVA instead is hopelessly slow in the more difficult cases, because it enumerates codes and not cubes and does not avoid the generation of symmetric encodings.
- The implicit algorithms of [3] rely on a very sophisticated unate covering package that represents the table with ZBDDs. MINSK instead is a simple-minded implementation, whose strength lies only in the underlying theory. The running times of MINSK can be improved a lot by making more efficient some critical routines such as *found_solution*.

6 Conclusions

We have presented a new matrix formulation of the face hypercube embedding problem that motivates the design of an efficient search strategy to find an encoding that satisfies all faces of minimum length. Increasing dimensions of the Boolean space are explored; for a given dimension constraints are satisfied one at a time. The following features help to reduce the nodes of the solution space that must be explored: candidate cubes instead of candidate codes are generated, symmetric solutions are not generated, a smaller sufficient set of solutions (producing basic sections) is explored, necessary conditions help discard unsuitable candidate cubes, early detection that a partial solution cannot be extended to be a global solution prunes infeasible portions of the search tree.

We have implemented a prototype package MINSK based on the previous ideas and run experiments to evaluate it. The experiments show that MINSK is faster and solves more problems than any available algorithm. Moreover, MINSK is a robust algorithm, while most of the proposed alternatives are not. All problems of the MCNC benchmark suite were solved successfully, except four of them unsolved or untried by any other tool. Other collections of examples were solved or reported for the first time, including an important set of decoder PLAs coming from the design of microprocessor instruction sets.

We know that the current implementation of MINSK is simple-minded and leaves room for improvements to speed-up more the program. For instance, the satisfaction check with the intersection matrix T_S is expensive and currently not optimized. Other areas of improvement to cope with difficult examples lie in the computation of a lower bound on the Boolean space dimension tighter than $\ln\ states$; and in a dynamic choice of the next cube and its size, based on a tighter analysis of the cube occupancy requirements of the existing constraints.

Moreover, we want to generalize the existing theory and algorithm in the following directions:

1. Solving face constraints with don't cares, that is an important practical problem.
2. Solving mixed problems that include constraints in the form of encoding dichotomies.

[2] An solution of 7 was erroneously reported as exact in [16] for *dk16*, whereas the minimum solution has 6 bits.

Name	#states	#cons.	min. len. / best known	min. sol.	#checks	#calls	time (secs)
bbsse	16	5	4 / 6	6	127	9	0.02
beecount	7	6	3 / 4	4	75	9	0.01
cse	16	9	4 / 5	5	219	11	0.03
dk14	7	9	3 / 4	4	137	16	0.02
dk15	4	6	2 / 4	4	66	12	0.01
dk16	27	24	5 / $\leq$ 8	6	622653	8686	161.45
dk17	8	7	3 / 4	4	162	9	0.01
dk27	7	4	3 / 3	3	42	5	0.00
dk512	15	9	4 / 5	5	569	12	0.06
donfile	24	24	5 / $\leq$ 6	6	245476	1722	48.14
ex1	20	8	5 / 7	7	1522	15	0.47
ex2	19	8	5 / 6	6	666	13	0.13
ex3	10	6	4 / 5	5	195	11	0.02
ex5	9	7	4 / 5	5	99	13	0.01
ex6	8	9	3 / 4	4	87	11	0.01
ex7	10	6	4 / 5	5	71	12	0.01
keyb	19	18	5 / 7	7	3676	184	1.62
kirkman	16	6	4 / $\leq$ 6	6	52	10	0.01
lion9	9	10	4 / 4	4	194	11	0.02
mark1	15	4	4 / 5	5	72	8	0.01
planet	48	10	6 / 6	6	2044	11	0.40
pma	24	13	5 / na	7	37339	687	14.42
s1	20	5	5 / 5	5	334	6	0.03
s1488	48	24	6 / na	-	-	-	timeout
s1494	48	24	6 / na	-	-	-	timeout
s208	18	5	5 / na	6	162	8	0.02
s27	6	6	3 / na	4	80	9	0.01
s298	218	47	9 / na				timeout
s386	13	5	4 / na	6	124	9	0.02
s420	18	5	5 / na	6	162	8	0.02
s820	25	10	5 / na	6	1832	13	0.38
s832	25	10	5 / na	6	1848	13	0.35
sand	32	5	5 / 6	6	131	7	0.02
scf	121	14	7 / $\leq$ 8	7	6239	17	2.82
sse	16	5	4 / 6	6	127	9	0.02
styr	30	16	5 / 6	6	973	18	0.29
tbk	32	73	5 / $\leq$ 18	-	-	-	timeout
tma	20	9	5 / na	6	2086	71	0.43
train11	11	11	4 / 5	5	8534	256	1.06

Figure 2: Experiments with FSMs from MCNC Benchmark Set.

From some preliminary analysis, both extensions are amenable to the current frame, with some appropriate modifications to the test when a candidate matrix is a solution and the introduction of the equivalent of a face for an encoding dichotomy.

Notice that the reduction of face hypercube embedding to satisfaction of encoding dichotomies [11, 3] has shown experimentally that face hypercube embedding in a sense contains the hardest instances of the problem to satisfy encoding dichotomies. This fact justifies our strategy to solve the former first and extend it later to the latter.

References

[1] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.

[2] O. Coudert. Two-level logic minimization: an overview. *Integration*, 17-2:97–140, October 1994.

[3] O. Coudert and C.-J. Shi. Ze-Dicho: an exact solver for dichotomy-based constrained encoding. In *The Proceedings of the International Conference on Computer Design*, pages 426–431, 1996.

[4] S. Devadas, A. Wang, R. Newton, and A. Sangiovanni-Vincentelli. Boolean decomposition in multilevel logic optimization. *IEEE Journal of solid-state circuits*, pages 399–408, April 1989.

[5] C. Duff. Codage d'automates et theorie des cubes intersectants. *Thèse, Institut National Polytechnique de Grenoble*, March 1991.

[6] E.I. Goldberg. Metody bulevogo kodirovaniya znachenij argumentov predicatov (methods of boolean encoding of predicate arguments values). *Preprint No. 3, Institute of Engineering Cybernetics, Academy of Sciences of Belarus*, 1991. (In Russian).

[7] E.I. Goldberg. Matrix formulation of constrained encoding problems in optimal PLA synthesis. *Preprint No. 19, Institute*

Name	ImpDicho time(s.)	ZeDicho time(s.)	Dicho time(s.)	MINSK time(s.)
dk16	spaceout	timeout	spaceout	161.45
dk512	timeout	timeout	238.72	0.06
donfile	timeout	timeout	spaceout	48.14
ex1	433.85	128.63	spaceout	0.47
ex2	timeout	timeout	spaceout	0.13
ex4	timeout	timeout	timeout	0.00
keyb	timeout	14.43	125.2	1.62
planet	spaceout	timeout	spaceout	0.40
s1	timeout	timeout	timeout	0.03
sand	spaceout	timeout	spaceout	0.02
scf	spaceout	timeout	spaceout	2.82
styr	spaceout	timeout	timeout	0.29
tbk	spaceout	timeout	spaceout	timeout

Figure 3: Comparison with Other Approaches.

Name	#states	#cons.	min. len.	min. sol.	#checks	#calls	time (secs)
apla	29	10	5	7	1267	14	0.45
lange	6	7	3	4	128	15	0.01
papa	7	9	3	4	162	19	0.02
scud	8	17	3	6	1102	77	0.28
tlc34stg	35	19	6	6	3635	20	1.22
viterbi	68	6	7	7	4510	7	1.07
vmecont	32	41	5	9	25958139	22354	95424.37
ir1a	128	2	7	8	22	4	0.01
ir1b	128	4	7	8	549	7	0.21
ir1c	128	5	7	8	1224	7	0.62
ir1d	128	11	7	9	402694	2299	512.10
ir2	128	8	7	8	10374	35	6.00
ir2m	128	11	7	8	31468	13	20.43
ir3	128	11	7	8	18075	13	13.89
ir4m	128	5	7	8	733	7	0.32

Figure 4: Experiments with FSMs from Other Sources (upper part of the table) and Decode PLAs of the VLSI-BAM (lower part).

of Engineering Cybernetics, Academy of Sciences of Belarus, 1993.

[8] E.I. Goldberg. Face embedding by componentwise construction of intersecting cubes. *Preprint No. 1, Institute of Engineering Cybernetics, Academy of Sciences of Belarus*, 1995.

[9] B. Holmer. A tool for processor instruction set design. In *The Proceedings of the European Design Automation Conference*, pages 150–155, September 1994.

[10] G. De Micheli, R. Brayton, and A. Sangiovanni-Vincentelli. Optimal state assignment for finite state machines. *IEEE Transactions on Computer-Aided Design*, pages 269–285, July 1985.

[11] A. Saldanha, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli. Satisfaction of input and output encoding constraints. *IEEE Transactions on Computer-Aided Design*, 13(5):589–602, May 1994.

[12] G. Saucier, C. Duff, and F. Poirot. State assignment using a new embedding method based on an intersecting cube theory. In *The Proceedings of the Design Automation Conference*, pages 321–326, June 1989.

[13] C.-J. Shi and J. Brzozowski. An efficient algorithm for constrained encoding and its applications. *IEEE Transactions on Computer-Aided Design*, pages 1813–1826, December 1993.

[14] J. Tracey. Internal state assignment for asynchronous sequential machines. *IRE Transactions on Electronic Computers*, pages 551–560, August 1966.

[15] T. Villa, T. Kam, R. Brayton, and A. Sangiovanni-Vincentelli. *Synthesis of FSMs: logic optimization.* Kluwer Academic Publishers, 1997.

[16] T. Villa and A. Sangiovanni-Vincentelli. NOVA: State assignment for optimal two-level logic implementations. *IEEE Transactions on Computer-Aided Design*, 9(9):905–924, September 1990.

[17] S. Yang and M. Ciesielski. Optimum and suboptimum algorithms for input encoding and its relationship to logic minimization. *IEEE Transactions on Computer-Aided Design*, 10(1):4–12, January 1991.

[18] A.D. Zakrevskii. *Logicheskii sintez kaskadnykh skhem (Logic synthesis of cascaded circuits).* Nauka, 1981. (in Russian).