

Solving Dichotomy-based Constrained Encoding With Twin Graph Coloring

Olivier Coudert
Synopsys, Inc., 700 East Middlefield Rd.
Mountain View, CA 94043

Abstract

Constrained encoding consists of finding a minimum length state encoding that meets some requirements. Dichotomy-based constrained encoding is more general than other constrained encoding frameworks, but it is also more difficult to solve in practice. This paper introduces a new formalization of this problem, which reduces it to what we call a *twin graph coloring*. This new problem leads to original exact and heuristic algorithms. Experimental results show that the exact solver is more efficient than the best known exact solvers, and that the heuristic version is competitive.

1 Introduction

One essential step in sequential logic synthesis consists of finding a state encoding that meets some requirements. These requirements include optimality criterions of the implementation (area and speed), and correctness criterions in the case of asynchronous finite state machines (race-free, hazard-free, or speed-independent implementation).

A dichotomy constraint forces some bit of the encoding to distinguish between two disjoint subsets of states (see Section 2.1 for a more formal definition). Tracey introduced dichotomy constraints when studying asynchronous finite state machine (FSM) state assignment [13]. He showed that if one chooses a state encoding satisfying some set of dichotomy constraints, then the resulting asynchronous implementation is critical race free. Finding a speed-independent asynchronous implementation comes down to dichotomy-based constrained encoding [14, 15]. State encoding targeting minimum-area PLAs is also related to dichotomy-based constrained encoding [7, 8, 9, 17], and more generally to optimal PLA implementations of Boolean expressions [3, 4]. State assignment for event-based specification is expressed as a constrained encoding problem in [6]. In [5] the problem of hazard-free minimization and asynchronous FSM encoding for multiple-input changes is reduced to dichotomy-based constrained encoding.

Fig. 1 illustrates the dichotomy based constrained encoding framework [11, 17]. Dichotomy constraints are used to express both the correctness and the optimization of the implementation. This framework is more general than some other frameworks, e.g. [16], which cannot

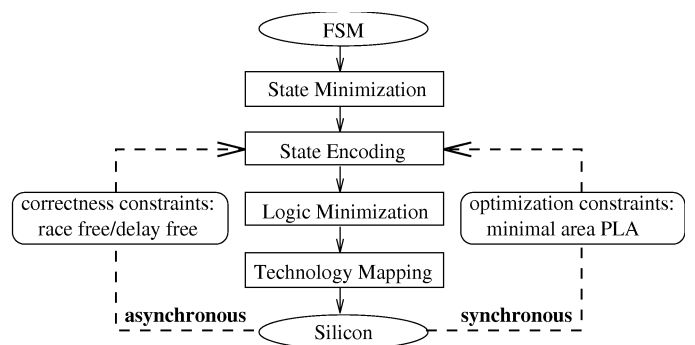


Figure 1: A unified framework for constrained encoding in sequential logic synthesis.

cope with the state assignment of asynchronous circuits.

This paper proposes a new paradigm to solve the dichotomy-based constrained encoding problem. Section 2 first introduces some definitions, and shortly discusses the expressiveness of dichotomy constraints. It then presents the state-of-the-art in solving dichotomy constraints. Section 3 introduces a new approach to dichotomy-based constrained encoding. It reduces it to what we call a *twin graph coloring*. It then gives some algorithmic details about how this new problem can be solved. Section 4 discusses some experimental results.

2 Terminology and Previous Work

This section presents some definitions, and discusses how the dichotomy-based constrained encoding problem has been attacked in the past.

2.1 Dichotomy Constraints

Let $S = \{s_1, \dots, s_n\}$ be a set of states. An *encoding* is a mapping¹ α from S into $\{0, 1\}^k$, where k is the length of the encoding. A *dichotomy* is an unordered pair $\{S_0, S_1\}$ of disjoint subsets of S . A dichotomy constraint expresses that the states of S_0 must be distinguished from the states of S_1 by at least one bit, i.e., this bit must have

¹It does not have to be injective.

	$\alpha(s)$		
1	0	0	0
2	1	0	—
3	0	1	1
4	1	1	0

Figure 2: A three-bit encoding.

the value 0 for all the states in S_0 , and the value 1 for all the states in S_1 , or vice versa. An encoding satisfies a dichotomy if and only if (iff) there is a bit of the encoding that meets the requirement expressed above.

Definition 1 *Given a set S of states and a set D of dichotomies, the constrained encoding problem consists of finding a minimum-length encoding that satisfies all the dichotomy constraints. This problem is NP-complete.*

For example², consider the four dichotomy constraints $\{\{1, 3\}, \{2\}\}$, $\{\{1\}, \{2, 4\}\}$, $\{\{1, 2\}, \{3, 4\}\}$, and $\{\{1, 4\}, \{3\}\}$, on the set of states $S = \{1, \dots, 4\}$. Fig. 2 shows a minimum-length encoding satisfying all these constraints.

Face hypercube embedding is another constrained state encoding framework [11]. A *facet* is a subset f of states of S that are constrained to be encoded in one face (cube) of the hypercube $\{0, 1\}^k$, without having any other state intersecting this face. E.g., if $f = \{1, 2, 3\}$, and with an encoding $1 = 1110$, $2 = 1010$, and $3 = 1100$, the face spanned by the facet f is $1 - -0$.

A set of facets can be easily translated to a set of dichotomies: for every facet f , generate the dichotomy $\{f, \{s\}\}$ with $s \in S - f$, which asserts that no other state intersects the face spanned by f ; also add the dichotomies $\{\{s\}, \{s'\}\}$ for $s, s' \in S, s \neq s'$, which asserts that every state is distinguishable. However, there is no systematic way to translate a set of dichotomies into a set of facets.

A dichotomy $\{S'_0, S'_1\}$ subsumes a dichotomy $\{S_0, S_1\}$ iff $S'_0 \supseteq S_0$ and $S'_1 \supseteq S_1$. Since any encoding satisfying some dichotomy d also satisfies the dichotomies d subsumes, all subsumed dichotomies can be removed from the set of constraints. A set of dichotomies is *irredundant* iff there is no subsumed dichotomy.

An *ordered* dichotomy (S_0, S_1) is an ordered couple of disjoint subsets of S . One says that an encoding satisfies an ordered dichotomy (S_0, S_1) iff there exists a bit of the encoding that has the value 0 for all the state of S_0 , and the value 1 for all the states of S_1 .

2.2 Previous Work

Tracey introduced the first exact³ algorithm for constrained encoding [13]. It is called *prime-covering* because of its similarity with the Quine-McCluskey procedure

²For the sake of simplicity, states are denoted with numbers in the examples given in this paper.

³Heuristic algorithms have been proposed in [13, 15, 17, 12].

for Boolean minimization, and it laid down the basis of most of the subsequent works.

A *compatible set* is a set of dichotomies that can be simultaneously satisfied by a single bit encoding. The *prime-covering* procedure [13] is as follows:

- (1) Build all the maximal compatible sets from the given set of dichotomy constraints D (in $O(\frac{3^{|D|}}{|D|})$);
- (2) Find a minimum number of maximal compatible sets that cover all the dichotomies (NP-complete).

A procedure that solves step (1), first suggested in [13] and described in [17, 9], is as follows. Two dichotomies are *compatible* iff they form a compatible set. Clearly, two dichotomies $\{S_0, S_1\}$ and $\{S'_0, S'_1\}$ are compatible iff there is no couple of states that are both in S_0 (or S_1) and that are splitted between S'_0 and S'_1 (and conversely, exchanging the prime). For example, $\{\{1\}, \{2, 3\}\}$ is compatible with $\{\{1, 4\}, \{2\}\}$, but not with $\{\{2\}, \{3\}\}$. A compatible set can be then represented by a new dichotomy, e.g., the compatibility of $\{\{1\}, \{2, 3\}\}$ and $\{\{1, 4\}, \{2\}\}$ is denoted with the new dichotomy $\{\{1, 4\}, \{2, 3\}\}$. Step (1) can be done by computing pair-wise compatible dichotomies, adding the new resulting dichotomies, and iterating this process until all compatible sets are maximal.

The prime-covering approach is seriously limited, because the number of maximal compatible sets generated by step (1) can be exponential w.r.t. $|D|$. However, what actually limits this approach to small size problems is the huge number of dichotomy pairs one needs to examine when generating the maximal compatible sets. In practice, dichotomies involve a small number of states, thus a lot of them are compatible. A set of m pair-wise compatible dichotomies (e.g., dichotomies whose subsets S_0 's and S_1 's are all mutually disjoint) produces $m(m-1)$ new dichotomies after one iteration. For k small, the number of dichotomies after k iterations is about m^{2^k} . One obtains about 10^8 dichotomies after 3 iterations from an initial set of 100 such dichotomies.

Note that a compatible set is necessarily made of pair-wise compatible dichotomies, but that the converse is false. For example, the three dichotomies $\{\{1, 2\}, \{\}\}$, $\{\{1, 3\}, \{\}\}$, and $\{\{2\}, \{3\}\}$, are pair-wise compatible, but they do not form a compatible set: the first two dichotomies force 2 and 3 to have the same encoding bit, while the third dichotomy requires these two states to have a distinguishing bit encoding.

The concept of compatibility extended to *ordered* dichotomies is more manageable, since a set of ordered dichotomies is compatible iff they are pair-wise compatible. Two ordered dichotomies (S_0, S_1) and (S'_0, S'_1) are compatible iff $S_0 \cup S'_0$ and $S_1 \cup S'_1$ are disjoint. Two (unordered) dichotomies $\{S_0, S_1\}$ and $\{S'_0, S'_1\}$ are compatible iff (S_0, S_1) is compatible with (S'_0, S'_1) or (S'_1, S'_0) . One associates two ordered dichotomies with each (unordered) dichotomy, and one obtains all the maximal compatible sets by first computing the maximal sets of compatible ordered dichotomies, and then converting the later into unordered dichotomies.

The *uncompatibility graph* (V, E) of a set V of ordered dichotomies is the graph obtained by linking with an edge two incompatible dichotomies of V . In [11] is presented an exact algorithm that computes the set of all maximal compatible sets from the uncompatibility graph (V, E) . Let us associate a Boolean variable with each vertex of the graph, and consider the following Boolean function:

$$f(v_1, \dots, v_n) = \bigwedge_{\{v, v'\} \in E} \neg(v \wedge v').$$

Any assignment that values f to 1 denotes a compatible set, made of the variables assigned to 1. Thus a maximal compatible set is the set of variables missing in some prime implicant of f . Since only two literals appear in each sum term, a linear-time algorithm for 2-SAT problem can be applied to convert the product-of-sums shown above to a sum-of-products representation. This leads to an explicit computation algorithm of all maximal compatible sets, whose complexity is linear w.r.t. the number of maximal compatible sets. However, as pointed out earlier, this method needs the explicit construction of all the maximal compatible sets, which can be exponential w.r.t. $|D|$.

In [2] is presented two ZBDD based algorithms to solve the constrained encoding problem. Both algorithms are based on set covering with compatible sets. Although ZBDDs allow to manipulate very large sets of compatible sets implicitly, these approaches suffer from the irreducibility of the resulting set covering problems.

3 Twin Graph Coloring

The methods that have been proposed so far to exactly solve dichotomy-based constrained encoding rely on building (explicitly or implicitly) the maximal compatible sets, then solving a covering problem. This section introduces a formalization that does no longer require the explicit notion of compatible sets, by reducing the original problem to a *twin graph coloring*. It then explains how to solve this new problem.

3.1 Formalization

A simple (i.e., undirected and self-loop free) graph G is denoted with (V, E) , where V is its set of vertices, and E its set of edges. Given a set of vertices V' , we will use the notation $G - V'$ to denote the subgraph induced by $(V - V', E)$. When the context is not ambiguous, we will denote a subgraph by its set of vertices.

Definition 2 A twin graph is a pair (G, T) , where $G = (V, E)$ is a simple graph, and T is a matching⁴ on V . Each pair $\{v, v'\}$ of T is called a *twin couple*, and we say that v is the *twin* of v' . A vertex which belongs to some pair of T is a *twin vertex*.

An *instance graph* of a twin graph (G, T) is a graph $G - V'$, where V' is a set of twin vertices that does not

⁴A matching on a set of vertices is a set of edges that has no self-loop and such that no two edges have a common endpoint.

contain any twin couple. In other words, it is obtained by removing from G only one out of the two twin vertices of some twin couples.

Coloring a graph consists of assigning a color to every vertex such that there is no two vertices linked by an edge have the same color. Let us define a coloring of a twin graph (G, T) as a coloring of one of its instance graphs. The minimum coloring of a twin graph is the minimum cardinality coloring that can be obtained over all its instance graphs. In other words, it is a minimum coloring of G such that for each pair of twin vertices, only one of them needs to be colored, the other one being removed from the graph.

The twin graph (G, T) derived from a set of (unordered) dichotomies D is such that V is the set of ordered dichotomies, $G = (V, E)$ is the uncompatibility graph of the ordered dichotomies, and T is made of the couples $\{v, v'\}$ such that v and v' are the two ordered dichotomies resulting from a unique unordered dichotomy. From now on, we identify vertex and ordered dichotomy, edge and uncompatibility.

Theorem 1 *The minimum coloring of the twin graph derived from a set of dichotomies D gives the minimum encoding satisfying D .*

Proof. (sketch) An independent set of the uncompatibility graph G is a compatible set. Thus any k -coloring of the uncompatibility graph, which partitions V into k independent sets $\{I_1, \dots, I_k\}$, yields a k -encoding. However, a minimum coloring of the uncompatibility graph does not necessarily produce a *minimum* encoding. This is because each dichotomy generates two ordered dichotomies, and that only one of those needs to be put in the partition, i.e., needs to be colored. This is precisely the definition of coloring a twin graph. \square

As an example, consider the set of four dichotomies $\{\{2, 3\}, \{\}\}$, $\{\{1, 3\}, \{2\}\}$, $\{\{1, 2\}, \{3, 4\}\}$, and $\{\{4\}, \{\}\}$ on a set of four states $\{1, 2, 3, 4\}$. Fig. 3 shows on the left the uncompatibility graph G of the complete set of ordered dichotomies. Edges between twin vertices are shown with a bold line. The middle graph is the same graph optimally colored with 6 colors, assuming it is a “normal” graph. The graph on the right shows the optimal coloring of the *twin* graph with 3 colors. The twin vertices and their incident edges that have been removed are shown with dotted lines.

There is an edge between a pair of twin vertices because they cannot be compatible⁵. However this edge does not have any effect when coloring the twin graph, since only one vertex in every twin couple needs to be colored. Consequently, we can remove the edge between each pair of twin vertices without changing the minimum coloring of the twin graph.

At first sight the reader may think that finding a “normal” minimum coloring of the graph resulting from removing every edge between twin vertices gives also a min-

⁵The only twin vertices that are compatible with each other are actually identical to the empty dichotomy $(\{\}, \{\})$. Note that it is compatible with any other dichotomy.

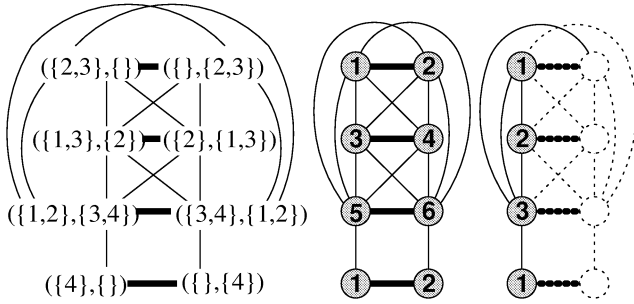


Figure 3: Dichotomies translated into a twin graph coloring.

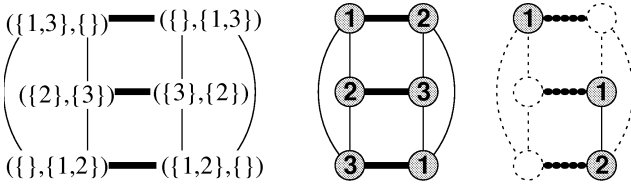


Figure 4: Color constraint propagation through symmetry breaking.

imum coloring of the twin graph derived from a set of dichotomies. One reason for this intuition is the strong symmetry of such twin graphs. Indeed, the twin graph derived from a complete set of symmetrized ordered dichotomies can be drawn as a graph whose pairs of twin vertices are organized in rows, and the graph is completely symmetric w.r.t. a vertical axis. In the example of Fig. 3, the minimum coloring of the graph in the middle, when ignoring the bold edges, is made of 3 colors, as is the minimum coloring of the twin graph. But this is not true in general. The reason is that the ordering of the dichotomies creates new edges that encode the *incompatibility* of sets made of otherwise *pairwise compatible* unordered dichotomies.

For instance, consider the three dichotomies $\{\{1, 3\}, \{\}\}$, $\{\{2\}, \{3\}\}$, and $\{\{\}, \{1, 2\}\}$. As we have seen in Section 2.2, they are pairwise compatible, but the three together does not form a compatible set. Fig. 4 shows the corresponding twin graph. The middle graph shows the same incompatibility graph, and its “normal” minimum coloring is made of 3 colors, with or without keeping the edges between twin vertices. However, the minimum coloring of the twin graph, shown on the right, is made of 2 colors only.

An overview of the algorithm that solves dichotomy based constrained encoding with twin graph coloring is as follows:

- (1) Compute a set of irredundant, reduced, ordered dichotomies V from the set of unordered dichotomies D (in $O(|S| \times |D|^2)$).
- (2) Build the incompatibility relation on V to obtain the twin graph (in $O(|S| \times |V|^2)$).
- (3) Color the twin graph (NP-complete).
- (4) Build an encoding from the coloring (in $O(|V| \times |S|)$).

Step (2) and (4) are straightforward. We address some details of steps (1) and (3) in the sequel.

3.2 Reduced Set of Ordered Dichotomies

If one generates all ordered dichotomies from D , one obtains a complete symmetric twin graph of $2|D|$ nodes. However, one can break the symmetry of this graph by generating a reduced set of ordered dichotomies.

Every ordered dichotomy (S_0, S_1) expresses that some bit of the encoding will be set to zero for all states of S_0 , and to one for all states of S_1 . Since complementing any bit of the encoding still preserves the dichotomy constraints, one can force one state to have an all-zero encoding. This means that from a completely symmetrized set of ordered dichotomies, one can select one state of S , and reject all ordered dichotomies (S_0, S_1) such that $s \in S_1$.

Choosing such a state s that minimizes the number of resulting ordered dichotomies can be done in time linear w.r.t. the size of all dichotomies, i.e., $\sum_{d \in D} |d|$ (bound by $|S||D|$, but much smaller in practice).

3.3 Coloring a Twin Graph

Fig. 5 outlines an algorithm to color a twin graph. It is derived from a classical sequential graph coloring algorithm by including specific treatments for twin vertices. The main differences are that (1) we need to handle a non-sequential backtrack (not shown in Fig. 5) that takes care of twin vertices; (2) we now have two options at each recursion: pick a twin couple $\{v, v'\}$ and remove v' from the graph, thus selecting v as the twin representative; or pick a vertex (whose twin vertex, if any, has been discarded as twin representative) and color it.

Pruning recursion when coloring twin graphs is rather challenging. A partial coloring yields a partial encoding, and the uncolored dichotomies it satisfies can be removed from the graph, thus pruning useless recursions. The most important aspect is the lower bound. The initial clique used as a starting point for the coloring must be made of twin free vertices. Consequently the clique is very small in practice and does not help much. Recomputing dynamically a clique after some twin representatives have been chosen is costly and still fairly ineffective.

The performance of the algorithm is very dependent on the choice of the action taken at each step: do we decide to select a twin representative, and if yes, which one? Or do we decide to color a vertex, and if yes, which one? We now discuss the possible strategies and heuristics to decide which action should be carry out at each recursion.

3.3.1 Selecting a Vertex to Color

A good heuristic to select a vertex to be colored is the DSATUR algorithm [1]. It consists of picking the vertex that has the largest saturation number (i.e., the number of forbidden colors, which are the colors used by its neighbors), and in breaking ties with the largest degree in the uncolored graph. The idea is to choose the vertex that is the most “difficult” to color, and that propagates as many color constraints as possible.

3.3.2 Selecting a Twin Representative

Let us assume that we have a function g that estimates the hardness of coloring a vertex v (the greater the value of $g(v)$, the more difficult v is to color). Let $\{v, v'\}$ be a twin couple whose representative has not been chosen yet. Since one can choose either v or v' as a representative, one can estimate the hardness of coloring this twin couple with $\min(g(v), g(v'))$. Let *Represent* be the function that selects the representative of a twin couple t . A natural heuristic consists of choosing the twin representative of $\{v, v'\}$ as the vertex that minimizes g :

$$\text{Represent}(t) = \arg \min_{v \in t} g(v).$$

We want to pick the representative of the most difficult twin couple to color, which is:

$$\arg \max_{t \in T} g(\text{Represent}(t))$$

3.3.3 Strategy

There are two extreme strategies to alternate coloring and picking twin representative:

- (1) Strategy “color as late as possible”: choose all the twin representatives first, then color the resulting graph.
- (2) Strategy “color as soon as possible”: each time a twin representative is chosen, it is immediately colored.

Strategy (1) performs poorly for the following reasons. If one selects the “wrong” twin representative v from a twin couple $\{v, v'\}$ in the early stages of the recursions, one has to exhaust the search for an optimal coloring on a suboptimal instance graph before backtracking and considering v' as the twin representative. Moreover, there is no color constraint information to help the twin representative selection. Strategy (2) performs reasonably well. However, when dealing with twin graphs whose symmetry has been broken, strategy (2) colors all twin free vertices before selecting twin representatives. Strategy (2) can underperform when the twin graph is not symmetric (which is always the case in practice with reduced sets of ordered dichotomies) and twin couples are far harder to color than twin free vertices.

```

function SC(G,T);
C = a clique of G made of twin free vertices;
k = 0;
foreach v ∈ C {
    k = k + 1;
    color v with k;
}
return SCrec(G, T, k, |V(G)| + 1, |C|);

/* (G, T) is a twin graph partially colored, using k colors, */
/* and best is the minimum number of color found so far */
function SCrec(G, T, k, best, lb);
if lb ≥ best return best;
if G is entirely colored return k;
v = select an uncolored vertex;
if (v has a twin v' still in G) {
    /* select v as the twin representative */
    for w ∈ {v', v} {
        remove w from G;
        C = a clique of G made of twin free vertices;
        best = SCrec(G, T, k, best, max(|C|, lb));
        restore w;
    }
}
else {
    /* v is a twin free vertex to color */
    for (c = 1; c ≤ min(k + 1, best - 1); c = c + 1) {
        if (∀v' ∈ N(v), color(v') ≠ c) {
            /* for each non-conflicting color c */
            color v with c;
            best = SCrec(G, T, max(c, k), best, lb);
            uncolor v;
        }
    }
}
return best;

```

Figure 5: Coloring a twin graph.

The general strategy, which can alternate twin representative selection and twin free vertex coloring in any order, opens many alternative heuristics. The simplest of them merely merges the twin representative heuristic with the coloring vertex selection heuristic. It consists of computing the vertex v defined as:

$$\arg \max g(v) \begin{cases} \text{if } v \text{ is twin free, or} \\ \text{if } v = \text{Represent}(t) \text{ for some } t \in T \end{cases}$$

If the resulting vertex is twin free, we color it, else we select it as the twin representative of the twin couple it belongs to. We use this heuristics, where g is derived from the DSATUR heuristic. In practice, this strategy closely mimics strategy (2) but overcomes the problem mentioned above.

4 Experimental Results

The first set of examples comes from the literature on the synthesis of asynchronous FSMs. The aim is either a race-free [13], a delay-free [14], or a hazard-free [5] implementation. It takes less than a second to solve all these problems.

The second set of tests consists of 40 MCNC industrial examples. In our experiments, we used ESPRESSO-MV [10] to generate face-embedding constraints. We compared the algorithm presented in this paper with the previous best known exact dichotomy solvers, [11] and [2]. The results are summarized in Table 1. The CPU time includes reading the facets, translating them into a set of irredundant unordered dichotomies, them into a reduced set of ordered dichotomies, building the corresponding twin graph, solving its minimum coloring, and building the minimum constrained state encoding out of it.

The twin graph coloring paradigm consistently beats the two other methods, and it can solve problems that fail to terminate otherwise. Note that our method finds the first known exact solution for the long-standing *tbk* example (the minimum solution is 17, while the best known upper bound was 18). When limiting the number of recursions to 5000 backtracks, one obtains a heuristic solver, which finds the optimum solution in all but 4 cases.

5 Conclusion

This paper has introduced a new paradigm to solve dichotomy-based constrained encoding, by reducing it to coloring what we called a *twin graph*. It proposed an algorithm to solve this problem, and discussed the different strategies and heuristics that take place during the resolution. Experimental results show that this algorithm beats the best known exact dichotomy based constrained encoding solvers.

The new formalization of dichotomy-based constrained encoding opens new opportunities to tackle the problem, exactly and heuristically. In particular, sophisticated vertex and action selection in the twin graph coloring algorithm could lead to better, more robust, and faster constrained encoding methods. One important aspect that should be improved is the dynamic lower bound computation, which is costly and difficult on twin graphs.

References

- [1] D. Brélaz, “New Methods to Color Vertices of a Graph”, *Comm. of the ACM*, **22-4**, pp. 251–256, 1979.
- [2] O. Coudert, C.-J. Richard Shi, “Exact Dichotomy-based Constrained Encoding”, in *Proc. of ICCD’96*, Austin TX, Oct. 1996.
- [3] S. Devadas, A. R. Newton, “Exact Algorithms for Output Encoding, State Assignment, and Four-level Boolean Minimization”, *IEEE Trans. CAD*, **1-10**, pp. 13–27, Jan. 1991.
- [4] S. Devadas, A. R. Wang, A. R. Newton, A. L. Sangiovanni-Vincentelli, “Boolean Decomposition of Programmable Logic Arrays”, pp. 2.5.1–2.5.5 in *IEEE Custom Int. Cir. Conf.*, 1988.
- [5] R. M. Fuhrer, B. Lin, S. M. Nowick, “Symbolic Hazard-free Minimization and Encoding of Asynchronous Finite State Machines”, pp. 604–611 in *Proc. of IEEE/ACM Int’l Conf. on CAD*, Nov. 1995.
- [6] L. Lavagno, C. W. Moon, R. K. Brayton, A. L. Sangiovanni-Vincentelli, “An Efficient Heuristic Procedure for Solving the State Assignment Problem for Event-based Specification”, *IEEE Trans. CAD*, **14-1**, pp. 45–60, Jan. 1995.
- [7] G. D. Micheli, R. K. Brayton, A. L. Sangiovanni-Vincentelli, “Optimal State Assignment for Finite State Machines”, *IEEE Trans. CAD*, **4-3**, pp. 269–285, July 1985.
- [8] G. D. Micheli, “Symbolic Design of Combinational and Sequential Logic Circuits Implemented by Two-level Logic Macros”, *IEEE Trans. CAD*, **5-1**, pp. 597–616, Oct. 1986.
- [9] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.
- [10] R. L. Rudell, A. L. Sangiovanni-Vincentelli, “Multiple-Valued Minimization for PLA Optimization”, *IEEE Trans. CAD*, **6-5**, pp. 727–750, Sept. 1987.
- [11] A. Saldanha, T. Villa, R. K. Brayton, A. L. Sangiovanni-Vincentelli, “Satisfaction of Input and Output Encoding Constraints”, *IEEE Trans. CAD*, **13-5**, pp. 589–602, May 1994.
- [12] C.-J. Shi, J. A. Brzozowski, “An Efficient Algorithm for Constrained Encoding and its Applications”, *IEEE Trans. CAD*, **12-12**, pp. 1813–1826, Dec. 1993.
- [13] J. H. Tracey, “Internal State Assignment for Asynchronous Sequential Machines” *IEEE Trans. Elec. Comput.*, pp. 551–560, Aug. 1966.
- [14] S. H. Unger, “A Row Assignment for Delay-free Realizations of Flow Tables Without Essential Hazards”, *IEEE Trans. Elec. Comput.*, **17-2**, pp. 145–158, Feb. 1968.
- [15] S. H. Unger, *Asynchronous Sequential Switching Circuits*, John Wiley & Sons, Inc., 1969.
- [16] T. Villa, A. L. Sangiovanni-Vincentelli, “NOVA: State Assignment of Finite State Machines for Optimal Two-level Logic Implementation”, *IEEE Trans. CAD*, **9-9**, pp. 905–924, Sept. 1990.
- [17] S. Yang, M. J. Ciesielski, “Optimum and Suboptimum Algorithms for Input Encoding and its Relationship to Logic Minimization”, *IEEE Trans. CAD*, **10-1**, pp. 4–12, Jan. 1991.

FSM	Example			Dichotomy & Twin graph					CPU		
	S	F	min/k	D	V	E	T	#back	[2]	[11]	twin
<i>bbsse</i>	16	5	4/6	60	75	1082	27	28681	9.50	2.38	1.66
<i>bbtas</i>	6	1	3/3	9	13	26	4	0	0.01	0.02	0.03
<i>beccount</i>	7	6	3/4	18	22	167	6	19	0.02	0.03	0.03
<i>cse</i>	16	9	4/5	59	81	1612	28	425	1.84	15.46	0.07
<i>dk14</i>	7	8	3/4	25	28	321	7	0	0.01	15.43	0.03
<i>dk15</i>	4	5	2/3	10	12	43	2	0	0.01	0.01	0.03
<i>dk16</i>	27	23	5/6	368	632	54730	264	—	—	*	—
<i>dk17</i>	8	7	3/4	31	46	446	15	5	0.01	0.10	0.01
<i>dk27</i>	7	4	2/3	21	30	178	9	56	0.02	0.04	0.02
<i>dk512</i>	15	9	4/5	106	180	5013	74	112638	—	238.72	27.77
<i>donfile</i>	24	24	5/6	480	860	136207	380	—	—	*	—
<i>ex1</i>	20	8	5/7	71	114	1909	43	11	128.63	*	0.05
<i>ex2</i>	19	8	5/6	106	175	5108	70	7088	—	*	1.23
<i>ex3</i>	10	6	4/5	38	57	701	20	186	0.05	0.31	0.04
<i>ex4</i>	14	1	4/4	91	169	1872	78	0	—	—	0.07
<i>ex5</i>	9	7	4/5	31	29	408	5	11	0.02	0.08	0.02
<i>ex6</i>	8	9	3/4	22	8	196	0	0	0.02	0.04	0.03
<i>ex7</i>	10	6	4/5	36	40	643	11	12	0.02	0.13	0.04
<i>keyb</i>	19	18	5/7	99	150	5062	51	441	14.43	125.2	0.23
<i>kirkman</i>	16	6	4/6	50	80	1057	30	2947	n/a	n/a	0.27
<i>lion</i>	4	3	2/2	6	3	11	0	0	0.01	0.01	0.01
<i>lion9</i>	9	10	4/4	44	65	1073	21	0	0.01	0.24	0.03
<i>mark1</i>	15	4	4/5	77	117	1780	48	4324	1.00	23.43	0.42
<i>mc</i>	4	1	2/2	6	9	12	3	0	0.01	0.01	0.01
<i>modulo12</i>	12	4	4/4	66	121	1100	55	0	28679.3	21.05	0.04
<i>opus</i>	10	2	4/4	33	56	326	23	0	1.59	0.31	0.03
<i>planet</i>	48	10	6/6	767	1435	79751	668	4722	—	*	1.66
<i>s1</i>	20	5	5/5	131	240	3678	109	15466	—	—	3.53
<i>s1a</i>	20	11	5/5	131	240	3678	109	15466	—	—	3.42
<i>s8</i>	5	1	3/3	7	10	15	3	0	0.01	0.01	0.01
<i>sand</i>	32	5	5/6	363	693	16928	330	—	—	*	—
<i>scf</i>	121	14	7/7	5683	11003	1371017	5320	—	—	*	—
<i>shiftrg</i>	8	5	3/3	28	43	375	15	0	0.01	0.09	0.03
<i>sse</i>	16	5	4/6	60	75	1082	27	28681	19.21	2.34	1.70
<i>styr</i>	30	16	5/6	193	326	9039	133	1168	—	—	0.80
<i>tav</i>	4	?	2/2	6	12	24	6	0	0.01	0.01	0.01
<i>tbk</i>	32	73	5/17	994	1695	666442	701	635	—	*	22.75
<i>train11</i>	11	11	4/5	96	157	4383	61	7814	84.22	5.67	2.20
<i>train4</i>	4	4	2/2	8	10	27	2	0	0.01	0.01	0.01

|S| : #states of the FSM.
|F| : #facets.
min/k: $\lceil \log_2(|S|) \rceil$ /minimum constrained encoding length.
|D| : #irredundant dichotomy constraints derived from the facets.
|V| : #nodes (i.e., #reduced ordered dichotomies).
|E| : #edges (i.e., #uncompatible pairs of ordered dichotomies).
|T| : #twin couples.
#back: #backtracks.
CPU : CPU time in seconds on a 167 MHz UltraSparc Workstation with 96MB
("—" is more than 2h, "*" is out of memory).

Table 1: Experimental results.