A Recursive Paradigm To Solve Boolean Relations

David Bañeres
Univ. Politècnica de Catalunya
Barcelona, Spain

Jordi Cortadella
Univ. Politècnica de Catalunya
Barcelona, Spain

Mike Kishinevsky
Strategic CAD Lab, Intel Corp.
Hillsboro, OR USA

ABSTRACT

A recursive algorithm for solving Boolean relations is presented. It provides several features: wide exploration of solutions, parametrizable cost function and efficiency. The experimental results show the applicability of the method and tangible improvements with regard to previous heuristic approaches.

Categories and Subject Descriptors: B.6.3 [Hardware]: Logic Design - Design Aids; J.6 [Computer Applications]: Computer-aided engineering

Terms: Algorithms, Design.

Keywords: Boolean relations, decomposition, logic design.

1. INTRODUCTION

Flexibility in, logic synthesis can be expressed using different abstract methods like don't care conditions (DCs), Boolean Relations (BRs), Multiple Boolean Relations (MBRs), sets of pairs of functions to be distinguished (SPFDs) [11, 14].

Don't cares form the basis for minimization of incompletely specified functions (ISFs) and multi-level networks. Boolean Relations allows to capture more flexibility than ISFs. However, while minimization of ISFs is a unate covering problem, solving BRs is a binate covering problem and hence is significantly more difficult [11].

Fig. 1.(a) illustrates an example of a BR with two input and two output variables. It is a subset of $\mathbb{B}^2 \times \mathbb{B}^2$, where $\mathbb{B} = \{0,1\}$. The input point 10 is related to two different points $\{00,11\}$, and 11 is related to another pair $\{10,11\}$. The expressed flexibility for 10 and 11 is different. The latter can be captured by introducing a don't care into the range of output variables ($\{10,11\} \equiv \{1-\}$). The former, $\{00,11\}$, cannot be expressed with don't cares.

To solve a BR one needs to find a compatible multi-output function with minimum cost. Figures 1.(b-c) depict two functions that are compatible with the original BR.

Many problems in logic design can be reduced to BRs: Boolean matching techniques for library binding [1], FSM encoding [10], Boolean decomposition [6], etc. For example, given a cut in the network, the flexibility of the nodes at the cut can be expressed by a BR. E.g. if the cut contains two nodes y_1, y_2 that reconverge to an *AND* gate, and for a given primary vector the output of the *AND* gate must be 0, then the flexibility at y_1, y_2 is $\{00,01,10\}$. This paper illustrates the use of BRs for using sequential elements with

DAC'04, June 7–11, 2004, San Diego, California, USA.

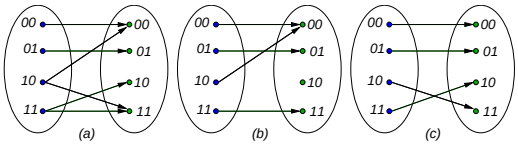


Figure 1: Example of Boolean relation (a) and two compatible functions (b,c).

embedded logic functions (e.g. a mux-latch) - a typical practice in high-performance design [8, 13], and for multi-way Boolean decomposition. The experimental results demonstrate that significant delay and area improvements can be achieved.

Different exact [2, 9] and approximate methods [7, 10, 16] for solving BRs have been reported. The exact methods can find a minimum solution provided that the cost function is the number of prime cubes. They are typically based on reducing the problem to the binate covering problem and solving it by Integer Linear Programming or by using reduction techniques and solving the cyclic core of the problem by branch-and-bound. The approximate methods use either testing techniques [7] or extend the iterative technique of ESPRESSO [16]. To the best of our knowledge, gyocro [16] is the most efficient solver so far. Therefore, we use gyocro as a reference for comparison in our experiments.

Our experience demonstrates that the number of cubes is not necessarily a good metric for estimating the complexity of solutions. Sometimes one needs to balance functions taking into account arrival times (for the delay) or balance the support of the functions (for reducing congestion in layout), etc. Our solver, BREL, is based on a recursive paradigm in which the cost function is a parameter. This allows to guide the search towards the user-defined goal. We also observed that gyocro cannot often escape from local minima determined by the initial solution, since the reduce-expand-irredundant loop of ESPRESSO is not always capable of hill climbing. In contrast, BREL uses a recursive paradigm implemented using branch-and-bound based on the following steps:

- Over-approximate the BR into a multi-output function
- Use standard methods for function minimization
- If the resulting ISF has no conflicts with the original relation, then report the result.
- Otherwise, select one conflict minterm.
- Decompose the original BR into two subBRs by taking different output components of the conflict minterm.
- Recursively solve the sub-BRs.

The rest of the paper is organized as follows. Section 2 presents basics of Boolean relations. Details of our solver are explained in Section 3. The major heuristics used to implement the recursive algorithm are presented in Section 4. Section 5 reports experimental results.

2. BACKGROUND

Definition 1. Boolean function. A Boolean function f is a function $f : \mathbb{B}^n \to \mathbb{B}$. A Boolean function can also be interpreted as the set of vertices $x \in \mathbb{B}^n$ such that $f(x) = 1$. An incompletely specified Boolean function (ISF) is a function $f : \mathbb{B}^n \to \mathbb{B} \cup \{-\}$, where $-$ is called the *don't care* value of the function. An ISF can be specified by three Boolean functions: $\mathsf{OFF}(f)$, $\mathsf{ON}(f)$ and $\mathsf{DC}(f)$ that characterize the vertices in $\mathbb{B}^n$ with image 0, 1 and $-$, respectively. □

An implementation of an ISF f is a Boolean function $\hat{f}$ such that

$$\mathsf{ON}(f) \subseteq \hat{f} \subseteq \mathsf{ON}(f) \cup \mathsf{DC}(f)$$

Definition 2. Cofactor and existential abstraction. The cofactors f_{x_i} and $f_{\overline{x_i}}$ of a Boolean function $f(x_1,\ldots,x_n)$ are defined as $f_{x_i} = f(x_1,\ldots,x_{i-1},1,x_{i+1},\ldots,x_n)$ and $f_{\overline{x_i}} = f(x_1,\ldots,x_{i-1},0,x_{i+1},\ldots,x_n)$. The existencial abstraction $\exists_{x_i} f$ is defined as $\exists_{x_i} f = f_{x_i} + f_{\overline{x_i}}$. Cofactors and existential abstraction can be extended to cubes. □

Definition 3. Boolean relation. A Boolean relation R is a subset of $\mathbb{B}^n \times \mathbb{B}^m$, where $\mathbb{B}^n$ and $\mathbb{B}^m$ are called the input set and output set of R, respectively. It can be specified by a characteristic function $R : \mathbb{B}^m \times \mathbb{B}^n \to \mathbb{B}$, such that $(x,y) \in R$ if and only if $R(x,y) = 1$. The complement of $R \subseteq \mathbb{B}^n \times \mathbb{B}^m$ is $\overline{R} = (\mathbb{B}^n \times \mathbb{B}^m) \setminus R$. A Boolean relation is *well defined* if for all $x \in \mathbb{B}^n$, there exists $y \in \mathbb{B}^m$ such that $(x,y) \in R$. □

Hereafter, we will indistinctively talk about sets and their corresponding characteristic functions. The union, intersection and complement of sets have dual operations with the disjunction, conjunction and complement of functions, according to Stone's representation theorem [15]. We will use $X = (x_1,\ldots,x_n)$ and $Y = (y_1,\ldots,y_m)$ to denote the set of inputs and outputs of a relation. We will also assume that Boolean relations are *well defined*, unless otherwise stated.

Definition 4. Multiple-output Boolean function. A well-defined Boolean relation $R \subseteq \mathbb{B}^n \times \mathbb{B}^m$ is a multiple-output function if

$$R(x,y_1) \wedge R(x,y_2) \implies y_1 = y_2$$

□

Definition 5. Compatible functions. Given a Boolean relation R, the set of multiple-output functions compatible with R is defined as

$$\mathbb{F}(R) = \{F \mid F \subseteq R \wedge F \text{ is a multiple-output function}\}$$

Note that $\mathbb{F}(R) = \emptyset$ if R is not well defined. □

Example 1. This example shows a tabular representation and the characteristic function of the Boolean relation that corresponds to Fig. 1(a).

x_1x_2	y_1y_2
00	{00}
01	{01}
10	{00,11}
11	{10,11}

$$R(x_1,x_2,y_1,y_2) = \overline{x_1}\,\overline{x_2}\,\overline{y_1}\,\overline{y_2} + \overline{x_1}x_2\overline{y_1}y_2 + x_1\overline{x_2}(y_1 \Leftrightarrow y_2) + x_1x_2y_1$$

In this example, the function

$$f(x_1,x_2,y_1,y_2) = (y_1 \Leftrightarrow x_1)\cdot(y_2 \Leftrightarrow (x_1 + x_2))$$

is compatible with the relation. However the function

$$g(x_1,x_2,y_1,y_2) = (y_1 \Leftrightarrow x_1)\cdot(y_2 \Leftrightarrow x_2)$$

is not compatible since it contains the vertex $x_1\overline{x_2}y_1\overline{y_2}$ that is not included in R. □

Definition 6. dc-extendable vertex. Given a Boolean relation R, its input vertex x is called *dc-extendable* if the output set corresponding to x can be captured as a single cube with don't cares. □

In the above example input vertex 11 is dc-extendable, while 10 is not. The following properties are important for the BREL solver.

Property 1. Semi-lattice of well-defined Boolean relations. The set of well-defined Boolean relations with the relation $\subseteq$ is a semi-lattice with one greatest element ($\mathbb{B}^n \times \mathbb{B}^m$) and 2^{n+m} least elements that correspond to $\mathbb{F}(\mathbb{B}^n \times \mathbb{B}^m)$. Let R and R' be Boolean relations and F be a function. Then,

$$\begin{array}{rcl} R \subset F & \implies & R \text{ is not well defined} \\ F \subset R & \implies & R \text{ is not a function} \\ R \text{ is well defined} & \implies & (R = R' \iff \mathbb{F}(R) = \mathbb{F}(R')) \end{array}$$

□

Definition 7. Projection of a Boolean relation. The projection of a relation $R(X,Y)$ onto the output y_i is another relation $(R \downarrow y_i)(X,y_i)$ defined by the characteristic function

$$(R \downarrow y_i)(X,y_i) = \exists_{y_1\ldots y_{i-1}y_{i+1}\ldots y_m} R(X,Y).$$

□

Example 2. From the relation presented in Example 1 the following projections can be derived:

$$\begin{array}{rcl} R \downarrow y_1 & = & x_1y_1 + \overline{x_1}\,\overline{y_1} + x_1\overline{x_2} \\ R \downarrow y_2 & = & x_1 + x_2y_2 + \overline{x_2}\,\overline{y_2} \end{array}$$

□

The projection of a relation onto one output implicitly defines an ISF for that output and represents the maximum allowable flexibility for that output still preserving the existence of a solution for the relation. The calculation of the ISF obtained from $R \downarrow y_i$ is denoted by

$$(\mathsf{ON},\mathsf{OFF},\mathsf{DC}) = \mathsf{ISF}(R \downarrow y_i)$$

Where the ON, OFF and DC set of the ISF are the following:

$$\begin{array}{rcl} \mathsf{ON}(R \downarrow y_i) & = & (R \downarrow y_i)_{y_i} \cdot \overline{(R \downarrow y_i)_{\overline{y_i}}} \\ \mathsf{OFF}(R \downarrow y_i) & = & (R \downarrow y_i)_{\overline{y_i}} \cdot \overline{(R \downarrow y_i)_{y_i}} \\ \mathsf{DC}(R \downarrow y_i) & = & (R \downarrow y_i)_{y_i} \cdot (R \downarrow y_i)_{\overline{y_i}} \end{array}$$

Example 3. For the previous example, we have

$$\begin{array}{rcl} R \downarrow y_1 & = & \underbrace{x_1x_2}_{\mathsf{ON}}y_1 + \underbrace{\overline{x_1}}_{\mathsf{OFF}}\,\overline{y_1} + \underbrace{x_1\overline{x_2}}_{\mathsf{DC}} \\ R \downarrow y_2 & = & \underbrace{\overline{x_1}\,x_2}_{\mathsf{ON}}y_2 + \underbrace{\overline{x_1}\,\overline{x_2}}_{\mathsf{OFF}}\,\overline{y_2} + \underbrace{x_1}_{\mathsf{DC}} \end{array}$$

where the ON, OFF and DC sets are

$$\begin{array}{rclrcl} \mathsf{ON}(R \downarrow y_1) & = & x_1x_2 & \mathsf{ON}(R \downarrow y_2) & = & \overline{x_1}x_2 \\ \mathsf{OFF}(R \downarrow y_1) & = & \overline{x_1} & \mathsf{OFF}(R \downarrow y_2) & = & \overline{x_1}\,\overline{x_2} \\ \mathsf{DC}(R \downarrow y_1) & = & x_1\overline{x_2} & \mathsf{DC}(R \downarrow y_2) & = & x_1 \end{array}$$

□

Given a set of single-output functions $F = \{f_i\}$, the characteristic function of the corresponding multiple-output function is obtained as follows:

$$F(X,Y) = \bigwedge_{i=1}^{m} (y_i \Leftrightarrow f_i(X))$$

Definition 8. Compatibility of a function. Given a function F and a relation R, F is *compatible* with R if $F \cdot \overline{R} = 0$. In general, we define

$$\mathsf{Incomp}(F,R) = F \cdot \overline{R}$$

as the set of vertices of F not compatible with R. □

Example 4. For the function $F = (y_1 \Leftrightarrow x_1) \cdot (y_2 \Leftrightarrow x_2)$ the incompatible vertices are $\mathsf{Incomp}(F,R) = x_1\overline{x_2}y_1\overline{y_2}$. □

Next, the basis of the divide-and-conquer approach presented in this paper is introduced.

Definition 9. Splitting a Boolean relation. Let $R \subseteq \mathbb{B}^n \times \mathbb{B}^m$ be a well-defined relation, $x \in \mathbb{B}^n$, and y_i one of the outputs of the relation. The following two relations can be defined:

$$R_1 = R \cdot (\overline{x} + \overline{y_i}); \quad R_2 = R \cdot (\overline{x} + y_i).$$

We denote the previous operation by

$$(R_1, R_2) = \mathsf{Split}(R, x, y_i).$$

□

Intuitively, given an input vertex x of the input set and one output y_i, the relation can be split into two relations such that one of them takes the value $y_i = 0$ and the other takes the value $y_i = 1$ for the vertex x. The two relations induce a partition over the functions compatible with R.

Property 2. Given R, R_1 and R_2 as defined above, the following properties hold:

$$R = R_1 \cup R_2; \quad \mathbb{F}(R) = \mathbb{F}(R_1) \cup \mathbb{F}(R_2); \quad \mathbb{F}(R_1) \cap \mathbb{F}(R_2) = \emptyset.$$

□

Property 3. If R is well defined and $(R \downarrow y_i)_x = 1$, then R_1 and R_2 obtained from $\mathsf{Split}(R, x, y_i)$ are well defined. □

Note that $(R \downarrow y_i)_x$ is a 1-variable function that can only be $\overline{y_i}$, y_i or 1. In the first (second) case, it indicates that the Boolean relation can only take the value $y_i = 0$ ($y_i = 1$) for the input vertex x. In the third case, it can take both values. It is easy to see that only in the third case, both R_1 and R_2 are well defined[1].

Example 5. Let us take the input vertex $x_1\overline{x_2}$ and the output y_1 from the relation in Example 1. Then, R_1 and R_2 are defined by the following tables:

R_1	
x_1x_2	y_1y_2
00	{00}
01	{01}
10	{00}
11	{10,11}

R_2	
x_1x_2	y_1y_2
00	{00}
01	{01}
10	{11}
11	{10,11}

Both R_1 and R_2 implicitly define a function for y_1 and an ISF for y_2. After minimizing each one, the functions

$$F_1 = (y_1 \Leftrightarrow x_1x_2) \cdot (y_2 \Leftrightarrow x_2)$$
$$F_2 = (y_1 \Leftrightarrow x_1) \cdot (y_2 \Leftrightarrow (x_1 + x_2))$$

are obtained, compatible with R_1 and R_2, respectively. Both solutions are compatible with R.

Note that if the vertex to split would be $\overline{x_1}\,\overline{x_2}$, then R_2 would not be well defined, since y_1 cannot take the value 1 for that vertex. □

[1]Note that $(R \downarrow y_i)_x = 0$ would indicate that R is not defined for vertex x and, therefore, is not well defined.

```
QuickSolver (R)
{Input: A well-defined relation R(X,Y)}
{Output: A multi-output function compatible with R}
   S := R;
   for each output y_i do
      (ON, OFF, DC) := ISF(S ↓ y_i);
      F_i := (y_i ⇔ Minimize(ON, OFF, DC));
      S := S · F_i;
   return S;
end;
```

Figure 2: A naive algorithm to solve a Boolean relation.

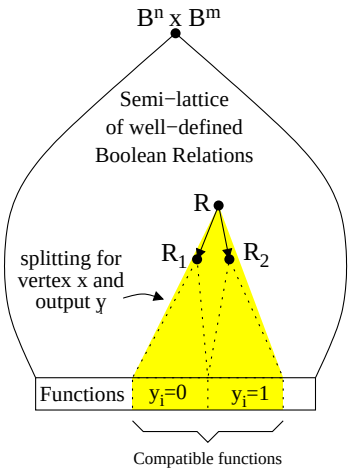


Figure 3: Recursive paradigm for solving Boolean relations.

3. BOOLEAN RELATION SOLVER

We first present a naive approach for solving Boolean relations. Next, the recursive algorithm is described.

3.1 Obtaining a quick solution

The algorithm from Fig. 2 was used in [16] to obtain an initial solution before applying the reduce-expand-irredundant process iteratively. It minimizes each output using the maximum flexibility provided by the relation. As long as outputs are calculated, the constraints of the solution are propagated to the rest of the outputs. The core of the algorithm is the function Minimize that performs the minimization of an ISF. Although this algorithm is efficient, it has two drawbacks:

- The solution depends on the order the outputs are minimized.
- The first outputs tend to take advantage of the flexibility of the relation, whereas the last outputs inherit little flexibitity. This leads to unbalanced and sub-optimal solutions.

The goal of this paper is to propose a method that performs a better exploration of the space of solutions, while having an affordable computational complexity.

3.2 The recursive approach

The approach proposed in this paper is based on the Split operation presented in Def. 9. The intuitive basis of this approach can be informally described as follows:

> *Let us minimize each output using the maximum allowable flexibility provided by the relation. If the obtained solution is not compatible, let us select a conflicting vertex and an output, and generate two relations compatible with the original one. They will cover the same*

```
BREL (R, BestF)
{Input: A well-defined relation R(X,Y) and the best
        found compatible function (BestF)}
{Output: BestF returns the minimum-cost function
         compatible with R}
  {Check for R to be a function}
  if R is a function then
    if cost(R) < cost(BestF) then BestF := R;
    return;

  {R is not a function}
  {Each output is minimized independently}
  F := 1;
  for each output y_i do
    F := F · (y_i ⇔ Minimize(ISF(R ↓ y_i)));

  if cost(F) ≥ cost(BestF) then return;

  {The solution is better, but it may not be compatible}
  I := Incomp(F,R);
  if I = 0 then BestF := F; return;

  {There are incompatibilities: split and call recursively}
  (x,y_i) := Pick (vertex, output signal) pair from I
             (x is not dc-extendable and (R ↓ y_i)_x = 1);
  (R_1,R_2) := Split(R,x,y_i);
  BREL(R_1,BestF);  BREL(R_2,BestF);
  return;
end;
```

Figure 4: A recursive algorithm to solve Boolean relations.

space of solutions, but will be more constrained (closer to the functions in the semi-lattice of Boolean relations). This produces an exploration tree in which the leaves correspond to the functions compatible with the relation.

The split operation is graphically illustrated in Fig. 3. The recursive algorithm is shown in Fig. 4. It is a branch-and-bound algorithm that uses the cost of the best explored solution to prune the search space.

It first detects when R is a function (terminal case). In case it is not, the minimization of functions with maximum flexibility is performed. The solution is rejected in case the cost is greater than the cost of the best obtained function so far. Note that the exploration is stopped in R if the solution is incompatible, under the assumption that constraining the relation to solve conflicts cannot improve the cost of a solution obtained by using the maximum flexibility. Finally, a vertex and an output are selected from the incompatible points.

This approach offers two main features:

- The cost function can be customized by the user, whereas the exact or heuristic solvers (e.g. [2, 16]) aim at minimizing the number of cubes of the solutions. Thus, the `cost` function in Fig. 4 can be a parameter of the recursive algorithm.
- The user can find a trade-off between the quality of the solution and the computational complexity spent in finding the solution. As in any branch-and-bound algorithm, the search can be aborted as soon as the resources (e.g. CPU time) have been exhausted.

The following property of the BREL algorithm implies that incompatible vertex flexibility cannot be captured with don't cares.

Property 4. If (x, y_i) is an incompatible vertex selected by the BREL algorithm, then x is not dc-extendable. □

In other words, incompatibilities may occur in this algorithm only at an input vertex x for which the output set cannot be precisely captured with don't cares. Consider example in Fig 1. BREL can potentially find an incompatibility for the input vertex 10, since its output set $\{00, 11\}$ cannot be captured with don't cares, but it would not consider 11 as an incompatibility conflict, since its output set $\{10, 11\}$ can be described as $1-$.

4. HEURISTICS AND IMPLEMENTATION ASPECTS

The general branch-and-bound approach presented in Fig. 4 can be relaxed and implemented in different ways. There are many degrees of freedom in implementing the approach: data structure to represent relations, strategy to explore the branch-and-bound tree, cost function, minimization of ISFs, etc.

We now present several implementation details of our solver, BREL, that lead to an efficient trade-off between the quality of the solutions and the computational complexity of the search. Most of the implementation decisions have been taken after experimenting with different strategies and choosing the most convenient.

4.1 Representation of relations

Binary Decision Diagrams (BDDs) are used to represent and manipulate the characteristic functions of the relations. All the transformations, evaluation of cost functions and minimizations of functions are performed with BDD operations.

The fact that all the relations generated by the solver come from a unique original relation tends to invoke many similar low-level BDD operations that are captured in the operation cache and calculated only once. This has an important impact in the performance of the solver.

4.2 Exploration of solutions

The branch-and-bound tree of solutions is explored by using a partial breadth-first-search (BFS). This requires a slight modification of the algorithm in Fig. 4. All the relations generated by splitting are stored in a list and visited in FIFO order.

The number of visited relations is also limited and is a parameter of the solver. For this reason, not all relations reach the leaves of the semi-lattice and become functions. To overcome this problem, the `QuickSolver` (Fig. 2) is used to guarantee a solution for each explored relation.

The BFS enables a wider diversity in the exploration of solutions and avoids the solver to spend his resources (CPU time) in only one corner of the tree seeking for a local optimum. Using hybrid approaches by combining BFS with DFS is left for future investigation.

4.3 Cost function

The cost function is a parameter of the solver. For efficiency reasons, BDD-based cost functions are desirable. Even though the size of BDDs is not always the best estimation of complexity for a Boolean function, typically there is a correlation among both. In the experimental results we have used different cost functions depending on the goal of the minimization: sum of BDD sizes when targetting at area minimization and sum of the square of BDD sizes when targetting at delay. The latter biases the exploration towards solutions in which the complexity of the functions is balanced, and hence the delay is more evenly distributed along all pathes. The former tends to minimized the overal size regardless of the relative complexity of sub-functions.

The experimental results show that these cost functions are suitable for the pursued goals.

4.4 Selection of a vertex with conflicts

When conflicts appear after the independent minimization of functions, an input vertex x and an output y_i must be selected for splitting. The solver uses the following strategy. Given the characteristic function of the conflicts, I, the outputs are existentially abstracted ($C = \exists_Y I$). Next, the shortest path (representing a large cube) in the BDD representing C is extracted. This cube characterizes a large region in $\mathbb{B}^n$ with conflicts. By constraining the value of the relation in one of the vertices of this region, many other adjacent vertices will tend to acquire the same value during minimization, due to the flexibility of the relation. This strategy accelerates the progress of the relation towards the bottom of the lattice (the functions).

We have experimentally found that this strategy is much more efficient than choosing the splitting vertex randomly.

4.5 Minimization of ISFs

This section refers to the Minimize operation in the solver. ISFs are defined by a pair of functions that represent the interval of flexibility $[\mathsf{Min}, \mathsf{Max}]$ (or $[\mathsf{On}, \mathsf{On} \cup \mathsf{Dc}]$). There are different methods to reduce the complexity of a function representation under the existence of flexibility. Some types of generalized cofactors, such as *constrain* and *restrict* [4,5], have been often used to reduce the size of BDDs. A BDD operation to find irredundant SOPs is also possible by using Minato-Morreale's algorithm [12], even though the obtained solutions may be far from the optimum.

Another way to reduce the complexity is to reduce the support by eliminating non-essential variables. A variable z is not essential if the interval $[\exists_z \mathsf{Min}, \forall_z \mathsf{Max}]$ is not empty (see [3], pp. 107–112).

Our solver first reduces the support of the ISF by greedily eliminating non-essential variables from top to bottom in the BDD. After that, an irredundant SOP is calculated by using Minato-Morreale's approach. We found this combined approach more efficient, in performance and quality of the solutions, than only using Minato-Morreale's minimization.

5. EXPERIMENTAL RESULTS

Table 1 presents comparative results with gyocro. In [16], a similar analysis is done for the exact minimizer [2] and Herb [7]. The number of cubes and SOP literals obtained by each solver are reported. Moreover, the size of the network after extracting common divisors (algebraic script in SIS) and technology mapping are also shown. The cost function used by BREL was the sum of BDD sizes for each output, aiming at minimizing area. The tree of solutions has been limited to the partial exploration of 10 Boolean relations in a breadth-first manner. Exploring more solutions did not contribute to improve results significantly.

The area results obtained by BREL are always better than gyocro (except for she1). Even though gyocro aims at minimizing the number of cubes, there are cases in which the solution obtained by BREL is significantly better (b9 and vtx). We attribute this phenomenon to the fact that gyocro may be trapped in a local minimum after generating the initial solution, from which it cannot easily escape by simply reducing and expanding cubes. On the other hand, the BFS strategy by BREL allows to explore a greater variety of solutions. The CPU time is usually better for BREL, with a tangible speed-up for two examples (b9 and vtx). However, the CPU times highly depend on the flexibility of the relation and on the number of reduce/expand/irredundant iterations performed by gyocro. A summary of the results is provided by the normalized sum shown at the last row.

Table 2 reports the results of an experiment designed to illustrate the applicability of BREL and the customization of its cost function. We consider the existence of a latch with an embedded mux in the library, and the next-state equation $Q = A \cdot C + B \cdot \overline{C}$. This three-input latch enables to implement the next-state function $F(X)$ as the composition of three functions: $A(X)$, $B(X)$ and $C(X)$. The Boolean relation specifying this flexibility is $F(X) \Leftrightarrow (A \cdot C + B \cdot \overline{C})$ where A, B and C are the output variables. The cost function has been defined as the sum of the squares of the BDD sizes for the three functions. The squaring favours a tendency to balance the complexity of the function and, therefore, reduce the delay of the circuit. In this case, 200 BRs in the BFS have been explored for each next-state function.

				ORIGINAL		Decomp. mux-latch		
	PI	PO	LT	Area	Delay	Area	Delay	CPU
daio	2	3	4	18096	3.47	11136	3.14	0.2
s27	4	1	3	15312	4.47	18096	4.01	0.2
ex2	3	3	19	347536	9.15	64496	6.39	0.3
ex3	3	3	10	169824	7.30	53824	5.50	0.1
ex5	3	3	9	144304	6.85	23664	5.02	0.1
ex7	3	3	10	178640	7.82	45008	5.35	0.1
s208	10	1	8	88160	7.58	98832	6.45	0.8
s298	3	6	14	134096	6.77	80736	4.48	1.7
s349	9	11	15	232000	9.45	284896	9.81	5.3
s382	3	6	21	225504	9.63	152192	6.07	3.1
s386	7	7	6	179104	7.84	152192	6.15	2.3
s420	18	1	16	207408	9.67	241280	9.73	24.6
s444	3	6	21	214832	8.64	169360	5.98	3.3
s510	19	7	6	300208	8.42	315520	8.05	203.6
s526	3	6	21	224576	8.75	169824	5.87	3.8
s641	35	23	19	522464	12.16	445904	9.28	7.4
s832	18	19	5	334080	10.53	375376	7.77	31.0
s953	17	24	29	507152	9.82	554480	8.11	35.0
s1196	14	14	18	1067664	11.87	1220784	12.62	5.9
s1488	8	19	6	741472	9.98	790192	9.57	10.2
s1494	8	19	6	729872	10.14	786016	9.59	10.2
sbc	40	56	28	920112	8.88	979504	8.73	20.3
Normalized sum				**1.00**	**1.00**	**0.92**	**0.84**	

Table 2: Logic decomposition for mux-latches (LT: number of latches)

The table reports the area and delay of the combinational part of the circuit[2]. The results have been obtained by collapsing the next-state functions, running the algebraic script[3], speed_up and technology mapping in SIS. For the mux-latch, the decomposition is done before running the algebraic script. In general, the results manifest several features of the approach: (1) the delay is usually reduced (sometimes significantly: e.g. ex2, s382, s641, s832), (2) in many cases area is also reduced due to the power of Boolean decomposition (e.g. ex2-7, s382, mult16b), (3) in some cases the delay is reduced at the expense of increasing area due to the balancing tendency of the cost function (e.g. s208, s510, s382, s953) and (4) the CPU time is affordable (only the CPU time of BREL is reported). In three cases (s349, s420 and s1196), both area and delay became worse with the mux-based decomposition.

The last row of the table summarizes the results and shows the global improvement obtained by the mux-based decomposition with Boolean relations.

6. CONCLUSIONS

A recursive paradigm for Boolean relations has been proposed. The results indicate that this approach is able to explore a greater diversity of solutions than previous heuristic approaches. The flexibility and efficiency of the algorithm makes it suitable for different

[2]For a more accurate apple-to-apple comparison, the set-up times and area of the simple and mux-latches should also be considered.
[3]The algebraic script is more convenient for delay optimization.

			gyocro					BREL				
	PI	PO	CB	LIT	ALG	AREA	CPU	CB	LIT	ALG	AREA	CPU
int1	4	3	5	8	8	9280	0.03	7	12	9	8352	0.01
int5	4	3	7	14	11	11136	0.02	7	14	11	11136	0.00
int10	6	4	25	88	32	44544	0.08	29	102	34	41296	0.02
c17b	5	3	7	12	12	10208	0.03	7	12	12	10208	0.00
c17i	5	3	15	37	34	34336	0.04	13	32	30	32016	0.01
she1	5	3	6	20	16	16240	0.04	9	26	15	17632	0.00
she2	5	5	10	33	31	30624	0.09	12	30	24	26448	0.01
she3	6	4	9	26	23	24592	0.08	9	27	21	21344	0.01
she4	5	6	20	91	56	62176	0.14	27	120	40	46864	0.03
gr	14	11	54	455	318	346608	3.43	86	590	313	322016	6.79
b9	16	5	270	2833	321	382336	4.68	137	1174	256	306240	0.19
int15	22	14	131	1083	506	525248	21.94	166	1062	459	472352	19.14
vtx	22	6	424	4460	117	151728	30.10	244	1809	101	94656	0.58
Normalized sum			**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.77**	**0.55**	**0.89**	**0.86**	**0.44**

CB: cubes, LIT: SOP literals, ALG: literals (after algebraic script), AREA: after tech. mapping (lib2.genlib)

Table 1: Comparison with gyocro [16].

applications. In the future, we foresee to explore different strategies of multi-way decomposition by using Boolean relations.

7. ACKNOWLEDGMENT

This work was supported by grants from Intel Corporation, CICYT TIC2001 2476, European Social Fund, FI and a Distinction for Research by the Generalitat de Catalunya.

8. REFERENCES

[1] L. Benini and G. D. Micheli. A survey of boolean matching techniques for library binding. *ACM Transactions on Design Automation of Electronic Systems*, 2(3):193–226, July 1997.

[2] R. Brayton and F. Somenzi. An exact minimizer for boolean relations. In *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, pages 316–319, Nov. 1989.

[3] F. Brown. *Boolean Reasoning: The Logic of Boolean Equations*. Kluwer Academic Publishers, 1990.

[4] O. Coudert, C. Berthet, and J. Madre. Verification of synchronous sequential machines using boolean functional vectors. In *Proc. IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, pages 111–128, Nov. 1989.

[5] O. Coudert and J. Madre. A unified framework for the formal verification of circuits. In *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, pages 126–129, Nov. 1990.

[6] M. Damiani, J. Yang, and G. D. Micheli. Optimization of combinational logic circuits based on compatible gates. *IEEE Transactions on Computer-Aided Design*, 14(11):1316–1327, Nov. 1995.

[7] A. Ghosh, S. Devadas, and A. Newton. Heuristic minimization of boolean relations using testing techniques. In *Proc. International Conf. Computer Design (ICCD)*, Sept. 1990.

[8] P. Hofstee and F. Alam. Design of high end processors for the consumer space, Nov. 2003. Talk at Sunday Workshop ICCAD.

[9] S. Jeong and F. Somenzi. A new algorithm for the binate covering problem and its application to the minimization of Boolean relations. In *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, pages 417–420, Nov. 1992.

[10] B. Lin and F. Somenzi. Minimization of symbolic relations. In *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, pages 88–91, Nov. 1990.

[11] G. D. Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw Hill, 1994.

[12] S. Minato. Fast generation of prime-irredundant covers from binary decision diagrams. *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, E76-A(6):967–973, June 1993.

[13] H. Partovi. Clocked storage elements. In A. Chandrakasan, W. Bowhill, and F. Fox, editors, *Design of High-Performance Microprocessor Circuits*, chapter 11, pages 207–234. IEEE Press, 2001.

[14] E. Sentovich and D. Brand. Flexibility in logic. In S. Hassoun and T. Sasao, editors, *Logic Synthesis and Verification*, chapter 3, pages 65–88. Kluwer Academic Publishers, 2002.

[15] M. Stone. The theory of representations for Boolean algebras. *Trans. Amer. Math. Soc.*, 40:37–111, 1936.

[16] Y. Watanabe and R. Brayton. Heuristic minimization of multiple-valued relations. *IEEE Transactions on Computer-Aided Design*, 12(10):1458–1472, Oct. 1993.

APPENDIX

A. EXAMPLE OF DECOMPOSITION WITH A MULTIPLEXOR

We provide an example to illustrate a function decomposition using a multiplexor. Let us assume that the following Boolean function is given:

$$f(x_1,x_2,x_3) = x_1(\overline{x_2}+\overline{x_3}) + \overline{x_1}x_2x_3$$

The goal is to decompose this function using a multiplexor and, therefore, to absorb part of the original function f within the multiplexer (Figure 5). The function for a multiplexor is $Q(A,B,C) = A\cdot C + B\cdot\overline{C}$. Hence, the characteristic function of the Boolean relation that captures flexibility of the implementation is

$$\begin{aligned} R(x_1,x_2,x_3,A,B,C) = f(x_1,x_2,x_3) \Leftrightarrow Q(A,B,C) = \\ (x_1(\overline{x_2}+\overline{x_3}) + \overline{x_1}x_2x_3)(AC + B\overline{C}) \\ +(x_1x_2x_3 + \overline{x_1}(\overline{x_2}+\overline{x_3}))(\overline{A}C + \overline{B}\,\overline{C}) \end{aligned}$$

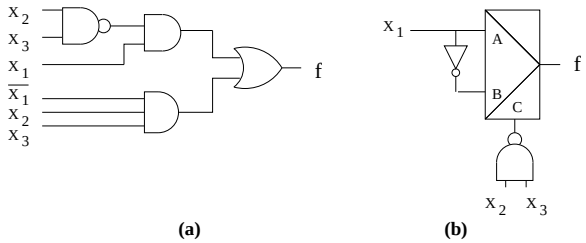
Our algorithm finds the following decomposition:

$$A(x_1,x_2,x_3) = x_1; \quad B(x_1,x_2,x_3) = \overline{x_1}; \quad C(x_1,x_2,x_3) = \overline{x_2}+\overline{x_3}$$

A tabular representation for the Boolean relation and the decomposition helps to illustrate the compatibility of the solution.

$x_1x_2x_3$	ABC
000	$\{-00, 0-1\}$
001	$\{-00, 0-1\}$
010	$\{-00, 0-1\}$
011	$\{1-1, -10\}$
100	$\{1-1, -10\}$
101	$\{1-1, -10\}$
110	$\{1-1, -10\}$
111	$\{-00, 0-1\}$

$x_1x_2x_3$	ABC
000	011
001	011
010	011
011	010
100	101
101	101
110	101
111	100



(a) **(b)**

Figure 5: (a) Original function. (b) Function decomposed using a multiplexor.