New Ideas for Solving Covering Problems

Olivier Coudert Jean Christophe Madre

Synopsys, 700 East Middlefield Rd.
Mountain View, CA 94040

Abstract

Covering problems occur at several steps during logic synthesis including two-level minimization and DAG covering. This paper presents a better lower bound computation algorithm and two new pruning techniques that significantly improve the efficiency of covering problem solvers. We show that these techniques reduce by up to three orders of magnitude the time required to solve covering problems exactly.

1 Introduction

The (unate or binate) covering problem is a well known intractable problem. It has several important applications in logic synthesis, such as two-level logic minimization, two-level Boolean relation minimization, three-level NAND implementation, state minimization, exact encoding, and DAG covering [1, 3, 2, 10, 7, 12, 15, 14, 8].

Although from the practical point of view people are more interested in low cost heuristic algorithms that produce approximate solutions, solving covering problems exactly is the only way to evaluate the performance of these heuristic algorithms. Moreover, improving exact solvers can bring out new effective heuristics algorithms: bounding the number of operations that an exact algorithm is allowed to perform is the simplest way to get a heuristic algorithm.

In practice one encounters covering problems whose covering matrices are small but that cannot be solved because the search space is too large. The only way of reducing the search space is to prevent the solver from exploring unsuccessful branches. This relies on two aspects: lower bound computation and pruning techniques.

This paper proposes new techniques that improve significantly the performance of covering problem solvers. The first one deals with computing a lower bound of the minimal solution of a covering problem. The lower bound computation algorithm which results is more costly than others that have been proposed, but it provides better lower bounds on difficult covering problems. The second improvement consists of two new pruning techniques. They are the most important from the practical point of view: first, they can be incorporated in *any* branch-and-bound based covering problem solver; second, they guarantee a low overhead in the worst case and can yield an exponential reduction of the search space.

This paper is organized as follows. Section 2 reviews concepts related to covering problems. Section 3 presents the skeleton of a branch-and-bound solver. Section 4 addresses the lower bound computation and proposes a better lower bound computation heuristics. Section 5 introduces two pruning techniques. Section 6 presents experimental evidences that demonstrate the effectiveness of the pruning techniques and the robustness of the lower bound computation on difficult covering problems.

2 Covering Problem

This section reviews the unate and binate covering problems. It also reviews concepts that will be used in the sequel. For the sake of simplicity, the resolution techniques discussed later will be focused on set covering problems, i.e., unate covering problems. These techniques extend to binate covering problems as well, and this section outlines how it can be done through a simple transformation.

2.1 Set Covering Problem

Let X be a set, and $Y \subseteq 2^X$. An element y of Y covers an element x of X iff $x \in y$. A subset Y' of Y covers a subset X' of X iff $X' \subseteq \bigcup_{y \in Y'} y$. Let $Cost$ be a cost

32nd ACM/IEEE Design Automation Conference ®

	y_1	y_2	y_3	y_4	y_5
x_1	1	1		1	
x_2		1	1		
x_3				1	
x_4		1			1
x_5	1		1		
x_6			1		1

Figure 1. A covering matrix.

function defined on Y, i.e., a function from Y into the set of positive real numbers. For any subset Y' of Y, we define $Cost(Y')$ as $\sum_{y \in Y'} Cost(y)$.

Definition 1 (Set covering problem) *The set covering problem $\langle X, Y, Cost \rangle$ consists of finding a minimal cost subset of Y that covers X.*

The covering matrix associated with the set covering problem $\langle X, Y, Cost \rangle$ has rows labeled with elements of X and columns labeled with elements of Y, such that the element $[x, y]$ of the matrix is equal to 1 iff y covers x. Assuming the same non zero cost for all columns, the minimal solutions of the covering matrix shown in Figure 1 are $\{y_2, y_3, y_4\}$ and $\{y_3, y_4, y_5\}$.

A covering matrix can be simplified using essentiality [10], dominance [9], partitioning, and Gimpel's reduction [4, 13]. We only remind here the notion of essentiality and dominance. The reader is referred to [1, 15, 3] for a complete review of reduction techniques.

A column y is essential if it is the only one that covers some x. An essential column obviously must belong to the minimal solution, so it can be removed from the covering matrix as well as all the rows it covers. A row x is dominated if there is another row $x' \neq x$ such that covering x' necessarily results in covering x. A column y is dominated if it covers a subset of the rows covered by some $y' \neq y$, and $Cost(y') \leq Cost(y)$. Dominated rows and columns can be removed from the covering matrix to produce a smaller matrix whose minimal solutions are minimal solutions of the original problem. Iterating this row and column removal produces a fixpoint, called the cyclic core of the covering matrix [11].

2.2 Binate Covering Problem

Let f be a Boolean function from $\{0,1\}^n$ into $\{0,1\}$. We note $v = (v_1, \ldots, v_n)$ a n-tuple of $\{0,1\}^n$. Let $Cost_k$ be a positive cost associated with the positive literal x_k. The cost of a n-tuple v is $\sum_{k=1}^{n} v_k . Cost_k$.

	x_1	x_2	x_3	x_4
$x_1 + x_3 + x_4$	1	2	1	1
$\overline{x_1} + x_2 + \overline{x_4}$	0	1	2	0
$x_2 + \overline{x_3} + x_4$	2	1	0	1
$\overline{x_2} + x_3 + \overline{x_4}$	2	0	1	0

Figure 2. A binate covering matrix.

	$\mathbf{x_1}$	$\overline{\mathbf{x_1}}$	$\mathbf{x_2}$	$\overline{\mathbf{x_2}}$	$\mathbf{x_3}$	$\overline{\mathbf{x_3}}$	$\mathbf{x_4}$	$\overline{\mathbf{x_4}}$	d
$x_1 + x_3 + x_4$	1				1		1		1
$\overline{x_1} + x_2 + \overline{x_4}$		1	1					1	1
$x_2 + \overline{x_3} + x_4$			1			1	1		1
$\overline{x_2} + x_3 + \overline{x_4}$				1	1			1	1
r_1	1	1							
r_2			1	1					
r_3					1	1			
r_4							1	1	

Figure 3. A modified binate covering matrix.

Definition 2 (Binate covering problem) *The binate covering problem (also called minimum cost assignment problem) consists of finding a minimal cost n-tuple that values f to 1.*

A binate covering problem can be described with a covering matrix. Let $\prod_{i=1}^{m} s_i$ be a product-of-sums representation of $f(x_1, \ldots, x_n)$. The covering matrix M of the binate covering problem is a matrix made of m rows labeled with the sums s_i, and of n columns labeled with the variables x_j. An element $M[i, j]$ of the matrix is equal to 1 if $(x_j \Rightarrow s_i)$, to 0 if $(\overline{x_j} \Rightarrow s_i)$, and to 2 otherwise. A n-tuple $v \in \{0,1\}^n$ covers a row s_i iff it exists a j such that $M[i, j] = v_j$. The binate covering problem consists in finding a minimal cost n-tuple that covers all the rows of the matrix.

A product-of-sums representation of the function $f(x_1, x_2, x_3, x_4) = x_2 x_3 + (x_1 \oplus x_4)$ is for example:

$$(x_1 + x_3 + x_4)(\overline{x_1} + x_2 + \overline{x_4})(x_2 + \overline{x_3} + x_4)(\overline{x_2} + x_3 + \overline{x_4})$$

The covering matrix built from this product-of-sums is shown in Figure 2. Assuming the same non zero weight for positive literals, the minimal cost true assignments of f are the 4-tuples $(1,0,0,0)$ and $(0,0,0,1)$. The binate covering problem reduction and resolution techniques generalize the ones outlined in Section 2.1. The reader is referred to [2, 5, 6] for more details.

We now show how a binate covering problem can be mapped on a modified covering matrix with additional entries. We associate with each column x_j of the matrix

a pair of columns labeled with $\mathbf{x}_j$ and $\overline{\mathbf{x}_j}$. A dummy row r_j only covered by the couple of columns $(\mathbf{x}_j, \overline{\mathbf{x}_j})$ is added for $1 \leq j \leq n$. A dummy null cost column d is added, covering only the first m rows of the original matrix. For example, applying this tranformation on the matrix shown in Figure 2 produces the matrix shown in Figure 3.

Now let us consider a set covering resolution algorithm whose column dominance is modified in the following way. The dummy column d must never be removed or selected, and it can dominate (in the sense defined in Section 2.1) any different single column. One can remove a single column c if it is dominated by the other column of the pair that c belongs to. A pair of columns $(\mathbf{x}_j, \overline{\mathbf{x}_j})$ dominates a pair of columns $(\mathbf{x}_k, \overline{\mathbf{x}_k})$ iff $\mathbf{x}_j$ covers all non dummy rows covered by $\mathbf{x}_k$, $\overline{\mathbf{x}_k}$ covers all non dummy rows covered by $\overline{\mathbf{x}_j}$, and $Cost_j \leq Cost_k$. A dominated pair of columns $(\mathbf{x}_k, \overline{\mathbf{x}_k})$ is removed from the matrix as well as the rows covered by $\overline{\mathbf{x}_k}$.

Note that if column $\mathbf{x}_j$ (respectively $\overline{\mathbf{x}_j}$) is selected for branching or because it is essential, then column $\overline{\mathbf{x}_j}$ (respectively $\mathbf{x}_j$) becomes dominated by the dummy column d, an so is removed from the matrix. This guarantees that one cannot have both $\mathbf{x}_j$ and $\overline{\mathbf{x}_j}$ in the solution.

A minimal cost solution of the binate covering problem can be built from the minimal cost solution S which is produced by the modified algorithm using the following transformation: if $\mathbf{x}_j \in S$, then $x_j = 1$, otherwise $x_j = 0$.

Although rewriting a binate covering problem into such a modified covering matrix is not the most efficient way to solve it, it shows that we can restrict ourselves to a unate covering matrix modulo some transformations of the resolution algorithm.

3 Solving Set Covering Problems

When solving a covering problem, it is first simplified using the reductions mentionned in Section 2.1. This eventually yields a covering matrix that cannot be further reduced. If it is empty, a minimal solution is built with the essential columns found during the reduction, and by recompositions if some Gimpel's reductions have been applied [4, 13].

If the reduced covering matrix $C = \langle X, Y, Cost \rangle$ is not empty, a branch-and-bound resolution must be performed. This consists of selecting a column y and generating two new set covering problems. The first one, which considers that y belongs to the minimal solution, is $C_l = \langle X - y, Y - \{y\}, Cost \rangle$. The second one, which excludes y from the minimal solution, is $C_r = \langle X, Y - \{y\}, Cost \rangle$. The two problems C_l and C_r are then recursively solved which produces the minimal solution of C.

```
1.  X' ← ∅;
2.  while X ≠ ∅ do {
3.      x' ← an element of X;
4.      X ← X − τ(x');
5.      X' ← X' ∪ {x'};
6.  }
7.  return X';
```

Figure 4. Greedy lower bound computation.

Heuristics to properly choose a branching column are discussed in [15, 1]. Defining for a row x

$$Weight(x) = \min_{\substack{y \in Y \\ y \ni x}} Cost(y),$$

we choose

$$y = \arg\max_{y \in Y} \frac{1}{Cost(y)} \sum_{x \in y} \frac{Weight(x)}{|\{y \in Y \mid y \ni x\}|}. \quad (1)$$

The more rows y covers and the less costly y is compared to the weights of the rows y covers, the more likely y is to be selected.

Since the search space is in $O(2^{|Y|})$, it is very important to have pruning techniques that prevent the solver from exploring unsuccessful branches. Let C be a set covering problem yielded at some point of the binary search tree. We note $C.min$ the cost of its minimal solution, and $C.path$ the cost of the path that yields C, i.e., the sum of the costs of all y's that constitute the solution being built. We note $C.upper$ the global upper bound, i.e., the cost of the best global solution found so far. Let $C.lower$ be a lower bound of $C.min$. As soon as $C.path + C.lower \geq C.upper$, the branch C belongs to can be pruned.

4 Better Lower Bounds

It is critical to provide an accurate lower bound to terminate useless searches as early as possible. This section discusses new techniques to compute better lower bound in a reasonable amount of time.

For a covering matrix $C = \langle X, Y, Cost \rangle$, let $\tau(x)$ be $X \cap \bigcup_{y \ni x} y$. Let X' be an independent set of X w.r.t τ, i.e., a subset of X such that any two different rows x'_1 and x'_2 of X' satisfies $x'_1 \notin \tau(x'_2)$ and $x'_2 \notin \tau(x'_1)$. To cover X we must at least cover X', so a lower bound of $C.min$ is the minimum cost needed to cover X', i.e.,

$$\sum_{x' \in X'} Weight(x').$$

The problem is that finding an independent subset X' that maximizes this lower bound is NP-hard. This is why the computation of X' is done in a greedy way, as shown in Figure 4. The quality of the lower bound obtained with such a greedy algorithm depends on the heuristics that selects the row x' to be put in the independent set X'.

4.1 Better Heuristic Selection

Once a row x' has been selected, $|\tau(x')|$ rows are removed from X. The greater $|\tau(x')|$ is, the smaller X becomes, the less candidates remain to build a large independent set. This is why it has been proposed to choose the row x' that minimizes $|\tau(x')|$ [14, 15, 1]. As explained below, it works even better if we break ties by selecting the row x' maximizing $\sum_{x \in \tau(x')} |\tau(x)|$.

Removing $\tau(x')$ from X decreases the size of $\tau(x'')$ for all x'' such that $\tau(x'') \cap \tau(x') \neq \emptyset$, which can contribute to a larger independent set. Thus a better selection heuristic must take into account the quantity $|\tau(x)|$ for $x \in \tau(x')$: the greater it is, the less likely x belongs to a large independent set, and the more rows x'' will have this quantity decreased. This explains why the tie breaker introduced above is effective. Also, we must take into account the weight of x': the greater, the better. To capture these different effects, we select

$$x' = \arg\min_{x' \in X} \frac{1}{Weight(x')} \sum_{\substack{x \in \tau(x') \\ x \neq x'}} \frac{Weight(x)}{|\tau(x)|}. \quad (2)$$

This heuristic is much more costly than the one mentioned above, but experimental results show that it is a good balance between the computational cost and the quality of the lower bound that results.

4.2 Covering Matrix Reduction

Line 4 of Figure 4 removes $\tau(x')$ from X. This is equivalent to removing from the covering matrix all the columns y's covering x', as well as the rows these columns y's cover. After this operation, new dominated columns may have been created, and we can consequently simplify the covering matrix, as it is done for a cyclic core computation (see Section 2.1). During this reduction, all essential rows are put in the independent set and the terms $|\tau(x)|$ are updated for all remaining x's, which contributes to a better selection.

This reduction has a large overhead compared with the cost of selecting a row x', but it eliminates rows that cannot produce a larger lower bound.

5 Two New Pruning Techniques

The "standard" pruning technique consists in aborting a recursion as soon as one finds a lower bound that is greater or equal to the cost of the best solution found so far. This section presents two new pruning techniques. They can be easily incorporated in any covering problem solvers, and reduce dramatically the number of reductions and consequently the time consumption.

5.1 Pruning with C_l's Lower Bound

Theorem 1 *Let C be a set covering problem. If $C.path + C_l.lower \geq C.upper$, then both C_l and C_r can be pruned, and a* strictly *better lower bound for C is $C_l.lower$.*

If the lower bound of C_l satisfies the given condition, we do not even need to examine C_r[1]. In practice, pruning C_r thanks to C_l's lower bound does not occur very often, but sufficiently enough to reduce the number of recursions of about 5% on some examples. It happens when the selection heuristic fails to select a good branching column y. In the worst case, the only cost of this pruning technique is an arithmetic test at each branching.

5.2 The Limit Lower Bound

This section introduces the most important result. It allows us to reduce very large search spaces by several orders of magnitude.

Theorem 2 (Limit Lower Bound)
Let $C = \langle X, Y, Cost \rangle$ be a set covering problem and X' be an independent set of X w.r.t. τ. Let $C.lower = \sum_{x' \in X'} Weight(x')$ be the lower bound of C obtained thanks to X'. Let

$$Y' = \{y \in Y \mid y \cap X' = \emptyset, \; C.path + C.lower + Cost(y) \geq C.upper\}.$$

Then C can be reduced to $\langle X, Y - Y', Cost \rangle$.

An interesting feature is that when the limit lower bound applies, reducing $\langle X, Y, Cost \rangle$ to $\langle X, Y - Y', Cost \rangle$ makes in practice the recursion terminate immediately, i.e., the lower bound of $\langle X, Y - Y', Cost \rangle$ nearly always exceeds the global upper bound, or a better solution is immediately found in $\langle X, Y - Y', Cost \rangle$. Note that testing whether the limit lower bound applies has complexity

[1] The same result holds for binate covering problems, providing that the branching column y is unate. Checking this condition is no longer necessary if we use the modified resolution algorithm on the modified covering matrix presented in Section 2.2

$O(|X'|(|Y| - |X'|))$, which is not costly compared to the gain it can produce. The limit lower bound technique can only improve covering problem solvers.

One could object that under the condition given in Theorem 2, the problems C_l and C_r that are generated from C would be pruned in the "usual" way, i.e., by using a lower bound computation on C_l and C_r. It is not true: one would have to perform several branchings before increasing the lower bound and being able to prune the recursion. Moreover, one could have to select all possible columns, which yields to a $O(2^{|Y|})$ unsuccessful search space. Thus from the theoretically point of view, the gain produced by the limit lower bound is exponential. Evaluating the average gain is an open problem. Experimental results show that the limit lower bound can reduce the search space by three order of magnitudes, thus dramatically reducing the computational time.

6 Experimental Results

This section demonstrates the effectiveness of the ideas presented above. We study the resolution of difficult covering problems with ESPRESSO-EXACT [15] and four set covering solvers. These four solvers branch on the column satisfying equation (1) and break ties by picking randomly a column among the best candidates. They are identical but for the way they compute a lower bound and exploit pruning. The differences are described by Table 1. Solver **S1** is the closest to ESPRESSO-EXACT, but the branching column and independent set selection heuristics are different, plus the implementation use more recent techniques to manipulate the covering matrix.

Table 2 presents the experimental results obtained on examples that have been selected for their large search space. These examples consists in computing the minimal cost sum-of-products of some multi-output Boolean functions of the MCNC benchmark [16]. The cost function is the number of products, or some more complex cost function depending on some literals (in this case the name of the problem is postfixed with L or I).

Comparing **S1** and **S2** shows clearly the immediate gain produced by the new pruning techniques presented in this paper. Both the size of the search space and the CPU time are definitely in their favor.

The gain produced by the lower bound computation algorithm we proposed is intuitively less immediate, because it introduces a large overhead. The performances of SCHERZO show that reducing the matrix during the independent set computation often provides better lower bounds. The lower bound computed by this method can be up to 10% better compared to the one achieved by

	Lower bound		Pruning	
	Select	Red.	C_l's LB	LLB
S1	$\arg\min_{x' \in X} \lvert\tau(x')\rvert$			
S2	same		√	√
S3	equation (2)		√	√
Scherzo	same	√	√	√

Select : Heuristic selection (Section 4.1).
Red. : Reduction (Section 4.2).
C_l's LB: Pruning C_r with $C_l.lower$ (Section 5.1).
LLB : Limit lower bound (Section 5.2).

Table 1. The different solvers.

S1 and **S2**. This comes down to shortening the depth of search by a similar ratio, which can reduce exponentialy the search space. Note that because of its greater lower bound computational cost, SCHERZO is sometimes slower than **S2**, though the former's search space is always smaller than the latter's one. But SCHERZO is definitely the most robust solver. It is better than, or comparable to the other solvers, and it is the only one that succeed in solving problems *prom2*, *ex5*, and *max1024*[2].

7 Conclusion

Improving the efficiency of covering problem solvers has an immediate impact on logic synthesis tools. This paper proposed a new lower bound computation algorithm, and two pruning techniques. The lower bound computation algorithm has a large overhead compared to the usual heuristics, but it provides very good lower bound on difficult covering problems. The pruning techniques can be straightforwardly incorporated in any covering problem solver and can produce an exponential reduction of the search space. Experimental results demonstrate that the combination of these ideas yields an efficient and robust covering problem solver, which is up to three order of magnitudes faster than the state-of-the-art ESPRESSO-EXACT.

8 Acknowledgments

The author would like to thank Richard Rudell who confirmed with his own experimentations the effectiveness of the limit lower bound and who provided the experimental results for the covering solver of ESPRESSO-EXACT.

[2] We also improved the best known solution of some other MCNC benchmarks: *bench1* (122, previously 125), *test4* (100, previously 104), and *test3* (477, previously 491).

Name	row × col	sol	Espresso		S1		S2		S3		Scherzo	
			node	CPU	node	CPU	node	CPU	node	CPU	node	CPU
mlp4	303 × 313	121	3601	46.6	116	1.3	60	0.9	20	0.6	20	0.8
pdc	6550 × 18923	96	814	171.6	202	15.9	86	13.9	54	21.6	38	19.1
lin.rom	545 × 578	128	13431	1456.7	1348	46.1	382	16.3	152	12.5	24	4.0
m4L	392 × 621	101	300728	1554.6	35708	93.0	6282	28.3	4412	28.2	3958	37.0
addm4L	616 × 997	189	181092	2704.6	17596	150.1	1904	25.8	1260	25.0	1218	45.6
m2I	150 × 222	47	1190636	2870.0	137773	240.3	34208	94.3	27290	106.4	33232	187.6
foutL	152 × 427	40	873837	3665.6	148676	253.1	17740	53.1	17160	70.2	13828	89.4
mp2dL	458 × 398	30	12845	4216.3	8382	296.3	696	54.1	374	59.7	362	64.3
expsI	258 × 484	132	703460	4511.1	167576	1105.3	39771	343.0	26290	379.8	25112	737.3
test1	349 × 2075	110	738165	5078.3	34665	274.2	4655	58.4	500	10.7	316	13.9
mlp4L	418 × 587	121	3774328	61490.9	101836	625.2	11384	111.1	984	16.8	1016	27.2
max512L	419 × 514	133	9832465	83998.7	195469	1273.8	40052	405.1	18700	282.0	13086	258.8
ocex2	575 × 681	216	*na*	*na*	63797	1241.3	19906	491.5	4938	223.4	2212	136.7
ocex1	328 × 567	74	–	–	1869219	28788.5	120305	2696.2	13123	409.0	6681	431.5
max1024	917 × 904	259	–	–	–	–	–	–	–	–	173318	29453.0
ex5	687 × 974	65	–	–	–	–	–	–	–	–	105862	34282.5
prom2	1988 × 2617	287	–	–	–	–	–	–	–	–	26496	36350.3

row × col: size of the covering matrix the solver starts with.
sol : #columns of the minimal cost solution.
node : #nodes explored in the search tree.
CPU : CPU time in seconds on a 60 MHz SuperSparc (85.4 SpecInt).
– : solver stopped after 4 days of CPU.

Table 2. Comparison of set covering solvers.

References

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

[2] R. K. Brayton, F. Somenzi, "An Exact Minimizer for Boolean Relation", in Proc. of *ICCAD'89*, pp. 1062–1081, November 1989

[3] O. Coudert, "Two-Level Logic Minimization: An Overview", Integration, **17**-2, pp. 97–140, Oct. 1994.

[4] J. F. Gimpel, "A Reduction Technique for Prime Implicant Tables", in *IEEE Trans. on Elec. Comp.*, **14**, pp. 535–541, June 1965.

[5] A. Grasselli, F. Luccio, "A Method for Minimizing the Number of Internal States in Incompletely Sequential Networks", in *IEEE Trans. on Elec. Comp.*, **14**, pp. 350–359, June 1965.

[6] R. W. House, D. W. Stevens, "A New Rule for Reducing CC Tables", in *IEEE Trans. on Comp.*, **19**, pp. 1108–1111, November 1970.

[7] Z. Kohavi, *Switching and Finite Automata Theory*, McGraw Hill, 1978.

[8] K. Keutzer, "DAGON: Technology Binding and Local Optimization", in Proc. of *DAC'87*, pp. 341–347, June 1987.

[9] E. L. Jr. McCluskey, "Minimization of Boolean Functions", in *Bell Sys. Tech. Jour.*, **35**, pp. 1417–1444, April 1959.

[10] W. V. O. Quine, "A Way to Simplify Truth Functions", in *Am. Math. Monthly*, **62**, pp. 627–631, 1955.

[11] W. V. O. Quine, "On Cores and Prime Implicants of Truth Functions", in *Am. Math. Monthly*, **66**, pp. 755–760, 1959.

[12] J. Rho, G. Hachtel, F. Somenzi, R. Jacoby, "Exact and Heuristics Minimization of Incompletely Specified Finite State Machines", in Proc. of *EDAC'90*, Febrary 1990.

[13] S. Robinson, R. House, "Gimpel's Reduction Technique Extended to the Covering Problem With Costs", in *IEEE Trans. on Elec. Comp.*, **16**, pp. 509–514, August 1967.

[14] R. L. Rudell, A. L. Sangiovanni-Vincentelli, "Multiple Valued Minimization for PLA Optimization", in *IEEE Trans. on CAD*, **6**-5, pp. 727–750, Sep. 1987.

[15] R. L. Rudell, *Logic Synthesis for VLSI Design*, PhD thesis, UCB/ERL M89/49, 1989.

[16] S. Yang, *Logic Synthesis and Optimization Benchmarks User Guide*, Microelectronics Center of North Carolina, January 1991.