Cluster-Cover: A Theoretical Framework for a Class of VLSI-CAD Optimization Problems

C.-J. SHI
University of Iowa
and
J. A. BRZOZOWSKI
University of Waterloo

This article introduces a mathematical framework called cluster-cover. We show that this framework captures the combinatorial structure of a class of VLSI design optimization problems, including two-level logic minimization, constrained encoding, multilayer topological planar routing, application timing assignment for delay-fault testing, and minimization of monitoring logic for BIST enhancement. These apparently unrelated problems can all be cast into two metaproblems in our framework: finding a maximum cluster and finding a minimum cover. We describe paradigms for developing algorithms for these problems. First, a simple heuristic called greedy peeling is presented and characterized. We derive sufficient conditions that guarantee optimum solutions by greedy peeling. We generalize the performance analysis of a multilayer topological planar routing heuristic to greedy peeling for the general cluster-cover problems. We propose a performance bound of greedy set covering that can be computed efficiently for a given problem instance; this bound is much tighter than the previously known bounds. Second, prime covering—originally developed for logic minimization—is generalized to finding exact solutions for cluster-cover problems. Previously, only the connection between logic minimization and constrained encoding was known.

Categories and Subject Descriptors: B.6.3 [**Logic Design**]: Design Aids—*automatic synthesis*; *optimization*; *verification*; B.7.2 [**Integrated Circuits**]: Design Aids—*layout*; *placement and routing*; *verification*; B.7.3 [**Integrated Circuits**]: Reliability and Testing—*built-in tests*; *error-checking*; *test generation*; F.2.2 [**Theory of Computation**]: Analysis of Algorithms and

This work was supported in part by Natural Sciences and Engineering Research Council of Canada (NSERC) under Grant No. OGP0000871, by a grant from the Information Technology Research Centre of Ontario (ITRC), by an Ontario Graduate Scholarship, and by the U.S. Advanced Research Projects Agency (DARPA) under grant number F33615-96-1-5601 from the U.S. Air Force, Wright Laboratory, Manufacturing Technology Directorate.
The authors thank Steven Nowick of Columbia University for helpful discussions on exact hazard-free logic minimization, and anonymous reviewers for many useful comments.
Authors' addresses: C.-J. Shi, Dept. of Electrical and Computer Engineering, University of Iowa, Iowa City, Iowa 52242; email: ⟨cjshi@eng.uiowa.edu⟩; J. A. Brzozowski, Dept. of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1; email: ⟨brzozo@maveric.uwaterloo.ca⟩.

Problem Complexity—*nonnumerical algorithms and problems*

General Terms: Algorithms, Design, Performance, Verification

Additional Key Words and Phrases: Cluster-cover, logic minimization, NP-completeness, self-checking logic design, state assignment, topological routing

1. INTRODUCTION

Optimization problems arise in almost every phase of Computer-Aided Design (CAD) of Very Large Scale Integrated (VLSI) circuits and systems. To handle the ever-increasing design complexity and reduce the design cycle time, efficient and reliable algorithms are highly desirable. The design of such algorithms, however, has been a great challenge. Most VLSI-CAD optimization problems are not only large (involve millions of variables and constraints), but also inherently computationally difficult (i.e., NP-hard). Nevertheless, tremendous progress has been made in algorithm development for some of these problems. Perhaps the best example is two-level logic minimization [Brayton et al. 1984]. *Exact* algorithms have been developed; these algorithms are capable of finding optimum solutions reasonably fast for large problem instances [Coudert and Madre 1992; Lin et al. 1992; McGeer et al. 1993]. A natural question, and a primary motivation of this research, is whether it is possible to apply these techniques to other VLSI-CAD problems.

Besides exact solutions, many *heuristic* algorithms capable of finding good approximate solutions efficiently have been developed. The evaluation of both the efficiency and robustness of heuristic algorithms relies heavily on the use of benchmarks. Since benchmarks represent only a limited class of "real" instances, heuristics well tuned for benchmarks often turn out to be not very efficient for other practical instances. Hence, the analysis of the performance of various heuristics is practically important, although it is theoretically challenging. Recently, Cong et al. [1993] successfully derived performance bounds for a heuristic algorithm for multilayer topological planar routing. A natural question, and the second motivation of this research, is whether it is possible to generalize Cong's analysis of performance bounds for a multilayer topological planar routing heuristic to other VLSI-CAD heuristics.

A key observation of this work is that two-level combinational logic minimization and multilayer topological planar routing have the same combinatorial structure. We represent this structure in a new mathematical framework called *cluster-cover*. The class of VLSI-CAD optimization problems covered by this framework includes some apparently unrelated problems such as constrained encoding for the synthesis of sequential logic [Saldanha et al. 1991; Shi and Brzozowski 1993; Yang and Ciesielski 1991], application timing assignment for delay-fault testing [Iyengar and Vijayan 1992], and minimization of monitoring logic for built-in self-test (BIST) enhancement [Gössel and Jürgensen 1993].

The cluster-cover framework enables us to reuse two-level logic minimization techniques (*prime covering*) in other problems in the framework, and to generalize Cong's results to a class of VLSI-CAD heuristics (abstracted as the *greedy peeling* paradigm). Furthermore, the notion of cluster-cover relates to two mathematical concepts: set cover and matroid; this allows us to derive some theoretical results on greedy peeling heuristics and to demonstrate problem instances for which greedy peeling heuristics yield optimum solutions. Previously, only the connection between two-level logic minimization and constrained encoding has been observed [Tracey 1966]; a successful application of two-level logic minimization techniques to constrained encoding has been reported in Yang and Ciesielski [1991].

Some preliminary results of this article have appeared in Shi [1993], Shi and Brzozowski [1993] and Shi [1997]. The article is structured as follows. We introduce the cluster-cover framework in Section 2. Several related mathematical notions are described in Section 3. In Section 4, we describe a simple and efficient heuristic called greedy peeling for finding approximate solutions. The quality of the approximate solutions produced by greedy peeling is analyzed in Sections 5 and 6 for the maximum k-cluster and minimum α-cover problems, respectively. An exact approach called prime covering is described in Section 7. In Section 8, we show how five VLSI-CAD applications can be handled in our framework. Section 9 concludes the article.

2. CLUSTER-COVER FRAMEWORK

If E is a set, we denote the set of all subsets of E, called the *power set* of E, by 2^E. A *subset system* is a pair $(E, \mathcal{P})$, where E is a finite set, and $\mathcal{P}$, a set of nonempty subsets of E, is defined by an application-specific *predicate* $\Pi(P)$, where $P \in 2^E$. Thus $\mathcal{P} = \{P \subseteq E \mid \Pi(P)\}$; that is, a subset P of E is in $\mathcal{P}$ if it satisfies the predicate; that is, if $\Pi(P)$ is true. For a set E, we use $|E|$ to denote its size, that is, the number of elements in E.

Example 1. The subset system defined by $E = \{1, 2, 3\}$ and predicate $\Pi(P) : (\Sigma_{e \in P}\, e) < 4$ is $(\{1, 2, 3\}, \{\{1\}, \{2\}, \{3\}, \{1, 2\}\})$.

Example 2. The subset system defined by $E = \{1, 2, 3\}$ and predicate $\Pi(P) : (\Sigma_{e \in P}\, e) > 2$ is $(\{1, 2, 3\}, \{\{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\})$.

Subset systems with the following two properties are of particular interest.

A *subset-closed system* is a subset system satisfying the following two conditions:

(1) $\{x\} \in \mathcal{P}$ for any $x \in E$,
(2) $Y \in \mathcal{P}$, $X \subseteq Y$, and $X \neq \emptyset$ implies $X \in \mathcal{P}$.

A *superset-closed system* is a subset system satisfying the conditions:

(1) $E \in \mathcal{P}$,
(2) $Y \in \mathcal{P}$ and $X \supseteq Y$ implies $X \in \mathcal{P}$.

Properties (1) and (2) are called *nonemptiness* and *hereditary* properties, respectively. The subset systems in Examples 1 and 2 are subset-closed and superset-closed, respectively.

We now present an abstract framework, called *cluster-cover*.[1] We are given a set E, a predicate Π on the set of nonempty subsets of E that defines a subset-closed system $(E, \mathcal{P})$, and a predicate Γ on the set of subsets of $\mathcal{P}$ that defines a superset-closed system $(\mathcal{P}, \mathcal{C})$. The set E is called the *ground* set, and the elements in E are called *ground elements*. The subsets in $\mathcal{P}$ are called *clusters*, and the subsets in $\mathcal{C}$ are called *covers*. The predicate Π is called the *compatibility* predicate, and the elements in a cluster are said to be *compatible*. The predicate Γ is called a *coverability* predicate. In this article, we are interested only in a rather simple coverability predicate, which is defined as follows. Given α, $0 < \alpha \leq 1$, an *α-cover* is a set of clusters whose union contains at least the fraction α of the ground elements. So we can denote our cluster-cover framework simply by a triple (E, Π, α) (sometimes by $(E, \mathcal{P}, \mathcal{C})$).

We are interested in the following two optimization problems on a given cluster-cover structure. The *maximum k-cluster problem* is: given an integer k, find at most k clusters that together contain as many ground elements as possible. When $k = 1$, this becomes the *maximum cluster* problem. The *minimum α-cover* problem is: given a fraction α, find an α-cover that contains as few clusters as possible. When $\alpha = 1$, this becomes the *minimum cover* problem, or the *complete cover* problem.

We observe that since the predicate Π defines a subset-closed system, the previously defined minimum cover problem is essentially the *minimum partition problem*, where a *partition* is a cover whose clusters are pairwise disjoint. On the other hand, we observe that as far as the minimum cover problem is concerned, we can always change the subset system defined by Π from not-subset-closed into subset-closed. The idea is to introduce dummy subsets for subsets that are not in $\mathcal{P}$ but should be in $\mathcal{P}$ according to the hereditary property. If such a dummy subset is actually used in the solution, we simply replace it by the corresponding superset that exists in the framework. This transformation does not change our solution to either the minimum α-cover problem or maximum k-cluster problem.

Example 3. Suppose that the ground set E is $\{a, b, c\}$, and some application-specific predicate Π defines that the clusters are $\{a, b\}$, $\{b, c\}$, and all their subsets. An illustration of this structure is given in Figure 1. Note that the compatibility relation here is not transitive: $\{a, b\}$ and $\{b, c\}$ are clusters, but $\{a, c\}$ is not. Then, given $\alpha = \frac{2}{3}$, the set of α-covers is $\mathcal{C} = \{C_1, \ldots, C_5, \ldots\}$, where $C_1 = \{\{a\}, \{b\}\}$, $C_2 = \{\{a\}, \{c\}\}$, $C_3 = \{\{b\}, \{c\}\}$, $C_4 = \{\{a, b\}\}$, $C_5 = \{\{b, c\}\}$, and the remaining covers are all the supersets of the first five covers.

[1]We note that the concept of cluster has been used informally in many contexts, notably in netlist partitioning, to refer to a set of closely connected nodes. Also, the concept of cover has been used in logic synthesis [Brayton et al. 1984]. To the best of our knowledge, however, the formal framework of cluster-cover, as defined here, has not appeared before in the literature.

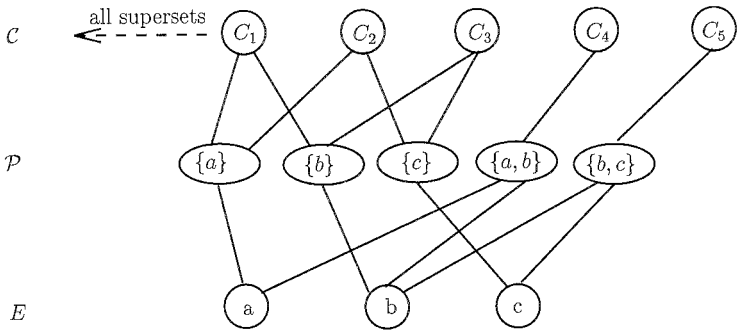

Fig. 1. Structure illustrating cluster-cover optimization problems.

3. RELATED MATHEMATICAL NOTIONS

Our aim in introducing the notion of cluster-cover is to provide a simple, convenient, and yet precise framework for capturing the underlying combinatorial structure of several VLSI-CAD optimization problems, while leaving out the problem-specific details. Several related mathematical notions have been used implicitly or explicitly in the literature to model cluster-cover problems in a number of VLSI-CAD applications.

3.1 Set Cover

Set cover is a notion familiar to VLSI-CAD researchers (due to the wide success of logic minimization). Given a ground set E, and a set $\mathcal{P}$ of subsets of E, a *set cover* is a collection of subsets from $\mathcal{P}$ that together cover all the elements in E. In our case the set of subsets is usually given implicitly by an application-specific predicate Π. In case the set $\mathcal{P}$, instead of its defining predicate Π, is given, cluster-cover is a direct extension of set cover: the minimum cover problem is the set cover problem.

3.2 Matroid

A subset-closed system $(E, \mathcal{P})$ that obeys the combinatorial aspect of the following Steinitz exchange principle is a *matroid* [Corman et al. 1990; Korte and Lovász 1984].

—If $X, Y \in \mathcal{P}$ and $|X| < |Y|$, then there exists a $y \in Y - X$ such that $X \cup \{y\} \in \mathcal{P}$.

The reason for noting this connection is the following. If the predicate Π in our cluster-cover framework defines a matroid, then the greedy algorithm will yield an optimum solution, as established in matroid theory [Corman et al. 1990] (See Section 4).

3.3 Partially Ordered Set

A *partially ordered set* is a special subset system (E, Π) where Π is a binary relation on E, which is

```
GREEDY_PEELING(E, k)
1  E' ← E
2  C ← { }
3  for i = 1 to ⌈k/h⌉ do
4     (P_i, E_i) ← MAX_h_CLUSTER(E')   /*solve*/
5     E' ← E' − E_i                    /*peel off*/
6     C ← C ∪ P_i                      /*augment*/
7  return (C, E')
```

Fig. 2. Paradigm of greedy peeling.

—*reflexive*, that is, $\Pi(x, x)$ for all $x \in E$;
—*antisymmetric*, that is, $\Pi(x, y)$ and $\Pi(y, x)$ imply $x = y$; and
—*transitive*, that is, $\Pi(x, y)$ and $\Pi(y, z)$ imply $\Pi(x, z)$.

A special case of the multilayer topological planar routing problem can be modeled as a problem on a partially ordered set; the latter problem can be solved in polynomial time [Cong et al. 1993].

3.4 Compatibility Graph and Conflict Graph

Suppose that a cluster cover $(E, \mathcal{P}, \mathcal{C})$ satisfies the following property, called *compatibility uniqueness*.

—Let $P \subset E$. If for all $X \subset P$ and $|X| > 1$, we have $X \in \mathcal{P}$, then $P \in \mathcal{P}$.

Then the compatibility predicate Π can be replaced by a simple binary *compatibility* relation $\mathcal{R}$ defined on E as follows. For any $x, y \in E$, $x \mathcal{R} y$ if $\Pi(\{x, y\})$ is true. Elements x and y are said to be *compatible* with each other.

We can represent a cluster cover satisfying the compatibility uniqueness property by either a compatibility graph or a conflict graph. This is done by introducing a node for each ground element. An edge joins two nodes in the *compatibility (conflict) graph* if the corresponding ground elements are compatible (incompatible).

The maximum cluster problem for a cluster-cover satisfying the compatibility uniqueness property can be stated as the maximum clique problem in its corresponding compatibility graph (the maximum independent-set problem in its corresponding conflict graph). The minimum cover problem is the minimum edge-covering problem with cliques (independent sets).

4. GREEDY PEELING

Greedy peeling is a well-known heuristic, which can solve both the minimum α-cover and the maximum k-cluster problems in a unified manner. It constructs directly the clusters needed in the final solution. Its formalization in terms of our framework is given in Figure 2. We have a ground set E and an integer k. The initial solution is an empty cover C (line 2). The core of greedy peeling is a subroutine called MAX_h_CLUSTER that solves the

maximum h-cluster problem exactly ($h \leq k$; typically $h = 1$ is used). The algorithm iterates $\lceil \frac{k}{h} \rceil$ times (lines 3 to 6). At each iteration, the subroutine is first invoked to find a set P_i of h clusters and the subset E_i of elements covered by P_i (step *solve*); then the elements covered by P_i are removed from E (step *peel-off*); and, finally, the newly found set P_i of clusters is added to the solution (step *augment*). The algorithm returns the computed cover C and the set of elements E' not covered by C.

THEOREM 1. *Greedy peeling yields an optimum solution for the maximum k-cluster problem, if the compatibility predicate Π satisfies one of the following properties.*

(1) *Transitivity: for any $x, y, z \in E$, $\Pi(\{x, y\})$ and $\Pi(\{y, z\})$ imply $\Pi(\{x, z\})$.*
(2) *Steinitz exchange property: if $X, Y \in \mathcal{P}$ and $|X| < |Y|$, then there is a $y \in Y - X$ such that $X \cup \{y\} \in \mathcal{P}$.*

Theorem 1 describes two cases where greedy peeling yields optimum solutions. (A proof is given in Appendix A.) In general, for problems with arbitrary compatibility predicates, greedy peeling may find only approximate solutions. One technique for improving the method is to prevent the peeling from being too greedy by imposing certain problem-specific criteria, for example, balance criteria in constrained encoding [Shi and Brzozowski 1993; Yang and Ciesielski 1991]. In fact, *almost-greedy peeling* may lead to results better than, or as good as, greedy peeling. This is an important observation, since the maximum h-cluster problem can be computationally expensive (NP-hard). We may choose to replace MAX_h_CLUSTER (step *solve*) by a simple and fast heuristic (called ALMOST_MAX_CLUSTER). Previously, we have applied a linear-time local search heuristic for ALMOST_MAX_CLUSTER to constrained encoding and obtained very encouraging results [Shi and Brzozowski 1993].

The second general technique for improving the solution quality of greedy peeling is to use iterative improvement. One can build local search on top of greedy peeling; this gives rise to *iterative greedy peeling*, which works as follows. Suppose we find a solution by greedy peeling after k iterations. The kth iteration was introduced, since there is a nonempty set E' of ground elements not covered by the first $k - 1$ iterations. The set E' thus appears to be hard to cover. Therefore, we start a new run of greedy peeling by insisting that the first iteration peel off E'. This technique has been proven to be very effective for constrained encoding [Shi and Brzozowski 1993].

5. GREEDY-PEELING PERFORMANCE FOR MAXIMUM k-CLUSTER PROBLEM

To analyze the performance of greedy peeling, we measure the nearness to optimality of a solution by the ratio of its greedy peeling solution (usually called *cost* for minimization problems) to that of an optimal solution. The performance of an approximation algorithm is then measured by its worst-case ratio over all inputs of a given size. For minimization problems, an

algorithm achieves ratio γ if for every problem instance it computes a solution whose cost is at most γ times the cost of the optimal solution. Here $1 \leq \gamma \leq \infty$. In the case of maximization problems, the value of the computed solution must be at least γ times the optimal value. Here $0 \leq \gamma \leq 1$.

We consider the performance ratio of greedy peeling for solving the maximum k-cluster problem. Our result here is inspired by the work of Cong et al. [1993] on the performance of a greedy heuristic for multilayer topological planar routing. We adapt their result to our cluster-cover framework and generalize it in two directions. First, we allow h clusters to be peeled off in one step, where $1 \leq h \leq k$. Second, we allow almost-greedy peeling with a performance ratio ϵ. Both of these generalizations make the result of Cong et al. [1993] more useful for practical applications. Mathematically, these generalizations are easily incorporated into the original proof (see Appendix B).

THEOREM 2. *Let* $r = \lceil \frac{k}{h} \rceil$. *Suppose that* MAX_$h$_CLUSTER *has performance ratio* ϵ. *Then the performance ratio of almost-greedy peeling for the maximum* k*-cluster problem is*

$$\gamma \geq 1 - \left(1 - \frac{\epsilon}{r}\right)^r.$$

Let $\gamma_0 = 1 - (1 - (\epsilon/r))^r$. We note that γ_0 has the following properties.

—When $r = 1$, $\gamma_0 = \epsilon$.

—If ϵ is a constant, then γ_0 is a decreasing function with respect to r. Since $\lim_{x\to\infty}(1 - \frac{a}{x})^x = (\frac{1}{e})^a$, $\gamma_0 \geq 1 - (1/e)^\epsilon$. Note that $0 < \epsilon \leq 1$. When $\epsilon = 1$, we have $\gamma_0 \geq 1 - \frac{1}{e} = 0.632$.

—$\partial\gamma_0/\partial\epsilon = (1 - \frac{\epsilon}{r})^{(r-1)}$. If $r = 1$, then $\partial\gamma_0/\partial\epsilon = 1$. *If* $r \to \infty$, then $\partial\gamma_0/\partial\epsilon = e^{-\epsilon}$.

—$\partial\gamma_0/\partial r = \epsilon/(r)^2(1 - (\epsilon/r))^r \ln(1 - \epsilon/r)$.

Theorem 2 establishes that greedy peeling has a guaranteed performance ratio γ (at least 0.632) for solving the maximum k-cluster problem. That is, the number of elements covered by k clusters found by greedy peeling is at least 63.2% of the number covered by the optimal solution.

6. GREEDY-PEELING PERFORMANCE FOR MINIMUM COVER PROBLEM

Johnson [1974] has derived $\ln|E| + 1$ as an attainable performance bound of greedy peeling for set covering. In this section, we describe a variant of the greedy peeling algorithm, which not only finds a greedy solution for set covering, but also computes a performance bound that is no greater than (and usually much smaller than) $\ln|E| + 1$ in practice. Our algorithm is based on an idea of Chvátal [1979].

To motivate the algorithm, we consider the following set cover example. Suppose $E = \{e_1, \ldots, e_8\}$ and $\mathcal{P}$ is a set of (explicitly given) clusters $P_1 = \{e_1, e_2, e_3\}$, $P_2 = \{e_1, e_4\}$, $P_3 = \{e_2, e_3, e_7\}$, $P_4 = \{e_3, e_5, e_7\}$, $P_5 =$

$\{e_3, e_6\}$, $P_6 = \{e_4, e_5, e_7\}$, and $P_7 = \{e_4, e_6, e_8\}$. We apply greedy peeling to find a complete cover of E.

Each time a cluster $P_i \in \mathcal{P}$ is peeled off by greedy peeling, we remove the elements in P_i from each of the clusters in $\mathcal{P}$. For convenience, we use $P_i^{(k)}$ to represent the cluster P_i after k iterations of greedy peeling. The set of nonempty clusters $P_i^{(k)}$ after k iterations of greedy peeling is denoted by $\mathcal{P}^{(k)}$. We use C to denote the cover. Initially $C = \{\ \}$ and $P_i^{(0)} = P_i$, $i = 1, \ldots, 7$. Since $P_1^{(0)}$, $P_3^{(0)}$, $P_4^{(0)}$, $P_6^{(0)}$, and $P_7^{(0)}$ all have the same largest size, greedy peeling arbitrarily picks one of them, say $P_6^{(0)}$, and adds it to the cover C. After removing the elements of $P_6^{(0)}$ from all the clusters in $\mathcal{P}^{(0)}$, we have the following set $\mathcal{P}^{(1)}$ of nonempty clusters. $P_1^{(1)} = \{e_1, e_2, e_3\}$, $P_2^{(1)} = \{e_1\}$, $P_3^{(1)} = \{e_2, e_3\}$, $P_4^{(1)} = \{e_3\}$, $P_5^{(1)} = \{e_3, e_6\}$, and $P_7^{(1)} = \{e_6, e_8\}$. Next, $P_1^{(1)}$ has the largest size, and will be added to the cover C. After peeling off $P_1^{(1)}$ from clusters in $\mathcal{P}^{(1)}$, there are two nonempty clusters: $P_5^{(2)} = \{e_6\}$ and $P_7^{(2)} = \{e_6, e_8\}$. We choose $P_7^{(2)}$ to add to the cover C. By now, all the elements of E have been covered, and we have obtained a cover $C = \{P_1, P_6, P_7\}$. Its cost is 3 clusters.

At each iteration, when a cluster, say $P_i^{(k)}$, is picked and added to C, the size of C increases by 1 cluster. We may say that the algorithm incurs a cost of 1. This cost can be spread evenly among the elements in the cluster $P_i^{(k)}$; that is, each element e_i in $P_i^{(k)}$ is assigned a cost of $c_i = [1/(|P_i^{(k)}|)]$. In our example, at the first iteration $P_6^{(0)}$ is peeled off, and each of the three elements e_4, e_5, e_7 in $P_6^{(1)}$ is assigned a cost $\frac{1}{3}$. At the second iteration, $P_1^{(1)}$ is peeled off. Each of three elements e_1, e_2, e_3 in $P_1^{(1)}$ is assigned a cost $\frac{1}{3}$. Finally, $P_7^{(2)}$ is peeled off. Since there are two elements, e_6 and e_8, covered in this iteration, each is assigned cost $\frac{1}{2}$. In summary, we have $c_1 = \frac{1}{3}$, $c_2 = \frac{1}{3}$, $c_3 = \frac{1}{3}$, $c_4 = \frac{1}{3}$, $c_5 = \frac{1}{3}$, $c_6 = \frac{1}{2}$, $c_7 = \frac{1}{3}$, and $c_8 = \frac{1}{2}$.

We can view these costs as weights associated with elements. Then the weight W_i of a cluster P_i can be calculated as the sum of the weights associated with all its elements. Note that the weights depend on the particular cover chosen by greedy peeling. In our example, we have $W_1 = 1$, $W_2 = \frac{2}{3}$, $W_3 = 1$, $W_4 = 1$, $W_5 = \frac{5}{6}$, $W_6 = 1$, and $W_7 = \frac{4}{3}$.

Let C' be an arbitrary cover that covers E; then

$$|\cup_{P_i \in C'} P_i| \geq |E|,$$

and

$$\sum_{P_i \in C'} W_i \geq \sum_{e \in E} c_e$$

$$= |C|.$$

Since an optimal cover, denoted by C^*, is one of these C', we have

$$\sum_{P_i \in C^*} W_i \geq |C|.$$

```
GREEDY(E, 𝒫, α)
 1  E' ← {}; C ← {}; k ← 0
 2  for P_i ∈ 𝒫 do
 3     W_i ← 0
 4  while |E'| < α|E| do
 5     k ← k + 1                          /* kth greedy peeling */
 6     P^(k) ← largest cluster in 𝒫        /*solve*/
 7     for P_i ∈ 𝒫 and |P_i| ≠ 0 do        /*peel off*/
 8        P_i ← P_i − P^(k)
 9        W_i ← W_i + |P_i ∩ P^(k)|/|P^(k)|
10     E' ← E' ∪ P^(k)
11     C ← C ∪ P^(k)                      /*augment*/
12  μ ← max{W_i | P_i ∈ 𝒫}
13  return (C, μ)
```

Fig. 3. Greedy peeling for minimum α-covering.

Thus,

$$|C| \le |C^*| \frac{\Sigma_{P_i \in C^*} W_i}{|C^*|}.$$

Note that $\gamma = (|C|/|C^*|)$ is the performance bound of greedy peeling. Hence,

$$\gamma \le \frac{\Sigma_{P_i \in C^*} W_i}{|C^*|}$$

$$= \text{average } W_i \text{ in } C^*$$

$$\le \max\{W_i | P_i \in \mathcal{P}\}.$$

This means that if we can compute a cover C by greedy peeling, then we know that the cost of an optimum cover C^* must be no less than $|C|$ divided by the maximum W_i such that $P_i \in \mathcal{P}$. In our example, the largest W_i is W_7, which equals $\frac{4}{3}$. So $\gamma \le \frac{4}{3}$, and $|C^*| \ge (|C|/\gamma) = \frac{9}{4} > 2$. Hence the greedy peeling solution C with cardinality 3 is proven to be optimum.

Note that W_i can be computed during the process of greedy peeling. This gives rise to an algorithm, called GREEDY, depicted in pseudocode in Figure 3. It is described for solving a more general problem, partial set covering: given a collection of subsets of elements in E, find a minimum number of subsets that cover at least the fraction α of elements in E. However, the value of $\max\{W_i | P_i \in \mathcal{P}\}$ computed by GREEDY may be no longer the upper bound for γ if $\alpha \neq 1$. To reflect this difference, we denote $\max\{W_i | P_i \in \mathcal{P}\}$ by μ.

THEOREM 3. *Let* $H(|P|_{\max}) = \sum_{i=1}^{|P|_{\max}} \frac{1}{i}$, *where* $|P|_{\max}$ *is the size of the first cluster picked by greedy peeling. Let* C *be the* α*-cover computed by greedy peeling, and let* C^* *be an optimum* α*-cover that covers the same set of elements as* C. *Let* μ *be computed by algorithm* GREEDY.

Then

$$\frac{|C|}{\mu} \leq |C^*| \leq |C|.$$

Furthermore,

$$\mu \leq H(|P|_{\max}).$$

A proof of Theorem 3 is given in Appendix C. It can be verified that $H(|P|_{\max}) \leq \ln|E| + 1$. When $\alpha = 1$, μ is the performance bound of greedy peeling. So we have the following corollary.

COROLLARY 1. *Let* C *be a complete cover computed by greedy peeling, and let* C^* *be an optimum cover. Then* μ *computed by algorithm* GREEDY *is a performance bound of greedy peeling; that is,*

$$\frac{|C|}{\mu} \leq |C^*| \leq |C|.$$

Further,

$$\mu \leq H(|P|_{\max}).$$

Noting that the proof of Theorem 3 itself does not need the set of clusters to be given explicitly, we have the following result.

COROLLARY 2. *The performance ratio* γ *of greedy peeling for the minimum cover problem satisfies*

$$\gamma \leq H(|P|_{\max}),$$

where $|P|_{\max}$ *is the size of the cluster picked by greedy peeling at its first iteration.*

In our example, $H(|P|_{\max}) = 1.83$, $\ln|E| + 1 = 2.46$, and $\mu = 1.33$. If we use either $H(|P|_{\max})$ or $\ln|E| + 1$ as a performance bound, we can only know that $|C^*| \geq 2$. These bounds do not indicate that $|C| = 3$ is the optimum, whereas the use of μ does.

We have conducted computational experiments on a set of randomly generated examples of the α-cover problem with $0 < \alpha \leq 1$. They all exhibit similar behaviors, one of which is illustrated in Figure 4.

For complete covering, an interesting question is whether we could find a better approximation algorithm, that is, an algorithm that achieves a ratio

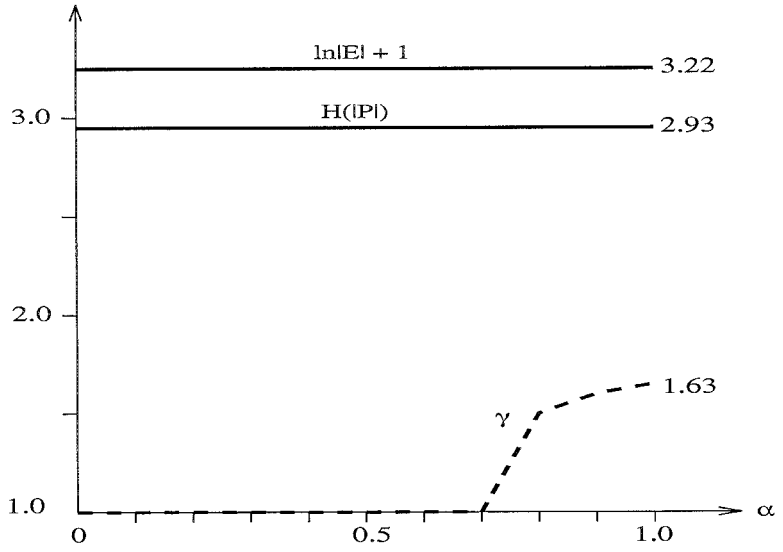

Fig. 4. A comparison of several bounds on an α-cover example.

smaller than that of greedy peeling. Unfortunately, Lund and Yannakakis [1993] recently discovered the following negative result: for any $0 < c < \frac{1}{4}$, the set-cover problem cannot be approximated within ratio of $c \log_2 |E|$ in polynomial time unless a very unlikely result holds in complexity theory; see Lund and Yannakakis [1993]. In view of this result, and the fact that $H(|P|_{\max}) \leq \ln|E| + 1 \approx 0.7 \log_2 |E| + 1$, greedy peeling is very likely the best-possible approximation algorithm for the optimization problems of the cluster-cover structure. This means that, although sophisticated heuristics may sometimes produce better results, they are very likely to have the same worst-case performance as simple greedy peeling.

7. PRIME COVERING FOR MINIMUM α-COVER

In this section, we present a branch-and-bound approach called prime covering for finding the optimum solution of the minimum α-cover problem. It consists of two stages, corresponding to the clusters and covers of Figure 1. First, the set of clusters is generated. Because it is subset-closed, only maximal clusters—for which the addition of one more ground element will violate the predicate—need to be found. Maximal clusters are called *primes*.

In the second stage, we solve a variation of the standard set-cover problem, where the number of elements to be covered is the fraction α of the total number of elements. The branch-and-bound method, in which the solution space is successively partitioned, can be used. A cluster is selected—this is called *branching*—and the problem is examined, first assuming that the cluster is in the minimum cover, and then that it is not. *Bounding* refers to generating lower bounds that can be used to prune the search space.

For complete covering, that is, $\alpha = 1$, we can use the greedy peeling algorithm GREEDY (Figure 3) to derive a tight lower bound for the objective function at each branch, and to compute a cover from which the next branching cluster will be selected. From Corollary 1, the number $|C|$ of prime clusters required to cover the fraction α of elements computed by GREEDY is at most μ (computed by GREEDY also) times that of an optimum solution. Therefore, the optimum solution is at least $|C|/\mu$. This can be used as a lower bound for the cost at each branch. Because $|C|$ is less than or equal to the optimum solution, the lower bound $|C|/\mu$ is at most μ times away from the optimum solution.

The choice of the branching cluster is based on the intuition that ground elements present in only a few clusters are hard to cover. To measure this hardness, each element is assigned a weight—the reciprocal of the number of clusters in which it appears. The weight of a cluster is the total weight of all its elements. The cluster that appears in the cover computed by greedy peeling and has maximum weight is chosen as the branching cluster.

Prime covering can be converted into a greedy heuristic by not using backtracking. The solution obtained is then an upper bound for the minimum α-cover.

The cluster-cover framework is defined using subsets of given sets. For practical applications, those subsets are sparse: each cluster contains only a few ground elements, and each cover contains only a few clusters. Therefore, a compact data structure called Zero-Suppressed Binary Decision Diagrams (ZBDDs), capable of manipulating the subsets very efficiently, can be used [Minato 1993]. This enables exact solutions of substantially large cluster-cover problems.

8. APPLICATIONS TO FIVE VLSI-CAD OPTIMIZATION PROBLEMS

In this section, we describe five applications and formulate them using the framework of cluster-cover. Among these applications, two-level combinational logic minimization and constrained encoding have been studied extensively and are well understood, multilayer topological planar routing and application timing for delay-fault testing are relatively new and less explored, and monitoring-logic design for BIST enhancement was introduced only very recently. The description is self-contained, and the emphasis is on the definition of clusters for each application.

8.1 Two-Level Hazard-Free Logic Minimization

A Boolean function f of n Boolean variables, $x_1, x_2, \ldots, x_n$, is defined as a mapping: $f : \{0, 1\}^n \rightarrow \{0, 1, *\}$, where $*$ represents a don't-care value (either 0 or 1). Each n-tuple in the domain $\{0, 1\}^n$ is called a *minterm*. The *on-set* (respectively, *off-set* and *don't-care set*) of f is the set of minterms for

which f has value 1 (respectively, 0 and $*$).[2] Boolean variable x_i and its negation $\overline{x_i}$ are called *literals*. A *product term* is a Boolean product (AND) of literals. If a product term evaluates to 1 for a given minterm, then the product term is said to *contain* that minterm. A *cube* is a set of minterms that can be described by a product term. A *cover* is a set of cubes (sum-of-products) such that every minterm in the on-set is contained in some cube, and no minterm from the off-set is contained in any cube. The basic *two-level logic minimization* problem is to find a cover that consists of a minimum number of cubes.

The two-level logic minimization problem can be formulated using the cluster-cover framework as follows. Let the on-set be the set of ground elements. A set of ground elements contained in a cube that does not have any minterms from the off-set forms a cluster; such a cube is an *implicant* of f. Then the two-level logic minimization problem is the complete cover problem in our framework.

Prime covering is a classical idea pioneered by Quine and McCluskey; it has been exploited extensively in recent years with *espresso* heuristics [Brayton et al. 1984; Rudell and Sangiovanni-Vincentelli 1987] and most recently with ZBDD-based techniques [Coudert and Madre 1992, 1995]. Therefore, our formalization does not contribute to new methods for this specific application. However, it is the very success of prime covering in logic synthesis that motivated us to extend the solution to a more general class of problems.

In the rest of this section, we show how the two-level hazard-free logic minimization problem, solved recently by Nowick and Dill [1995], can be solved exactly in our framework. Given a *start point* $A \in \{0, 1\}^n$ and an *end point* $B \in \{0, 1\}^n$, a *transition cube* from A to B, denoted by $[A, B]$, is the smallest cube that contains both A and B. It contains all the minterms that can be reached during a transition from A to B. A multiple-input change from minterm A to B is described by the transition cube $[A, B]$. For example, Figure 5(a) shows the Karnaugh map of the example used by Nowick and Dill [1995], where four input transitions are marked by $t1$ to $t4$. The four transition cubes are $a\bar{c}$, $a\bar{b}c$, $\bar{a}\bar{c}$, and c, respectively.

A transition from minterm A to B has a *function hazard* iff there is a *minimum-length path* from A to B along which the function value changes more than once. In this case, no implementation is guaranteed to avoid glitches [Nowick and Dill 1995; Unger 1969]. Therefore only function-hazard-free transitions are considered for hazard-free logic minimization. The four transitions in Figure 5(a) are function-hazard-free.

From the output perspective, there are four types of transitions: 1–1 transition, 0–0 transition, 0–1, and 1–0 transitions. For a transition from A to B, an *on-set subcube* S is such that $S \subseteq [A, B]$ and, for all the minterms X contained in S, $f(X) = 1$. Given a function-hazard-free

[2]Note that historically minterms for which f has value 0 are called maxterms [Kohavi 1978]. Here we follow a more recent convention due to Brayton et al. [1984], Nowick and Dill [1995], and Rudell and Sangiovanni-Vincentelli [1987].

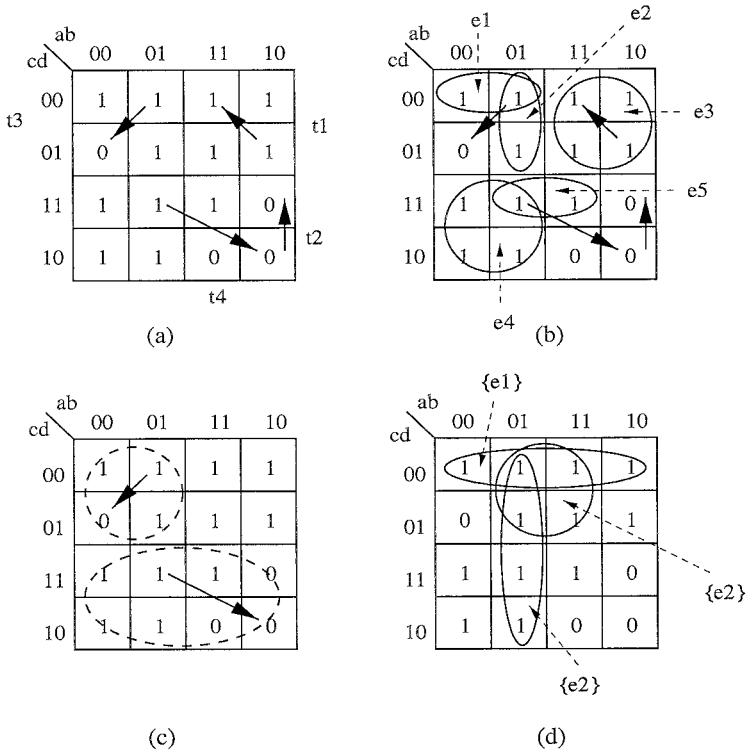

Fig. 5. Hazard-free logic minimization example: (a) Karnaugh map with input transitions; (b) ground elements; (c) 1–0 transition cubes; (d) from clusters to cubes.

transition, a set of *maximal* on-set subcubes can be derived *uniquely*. For example, consider transition $t4$ in Figure 5(a); its maximal on-set subcubes are $\bar{a}c$ and bcd. The maximal on-set subcube of a 0–0 transition is the empty cube; the maximal on-set subcube of a 1–1 transition is the transition cube itself. All the nonempty maximal on-set subcubes for the transitions in Figure 5(a) are illustrated in Figure 5(b); they can be presented uniquely by a set $E = \{a\bar{c}, \bar{a}\bar{c}\bar{d}, \bar{a}b\bar{c}, \bar{a}c, bcd\}$.

For a given set of function-hazard-free transitions, let the set of *required cubes* be the set of all the nonempty maximal on-set subcubes for all the transitions. Since only specified transitions are of interest, the set of required cubes thus defines the on-set of the function. According to Nowick and Dill, additional requirements for hazard-free two-level implementation are that every required cube be contained in some cube of the cover, and if a cube in the cover intersects a 0–1 or 1–0 transition cube, then it must contain its 1-end point.

To formulate the two-level hazard-free logic minimization problem using the cluster-cover framework, let the ground set be the set of required cubes. A cluster is a set of ground elements, which can be contained in a cube that does not intersect with the off-set, and if it intersects with a 0–1 or 1–0 transition cube, then it must contain its 1-minterm; such a cube is called a

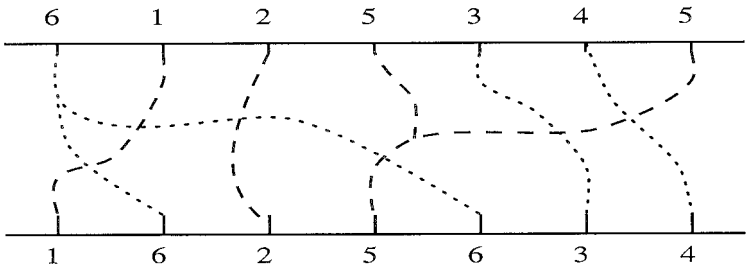

Fig. 6. Example of multilayer topological routing.

dhf(dynamic-hazard-free)-prime implicant in Nowick and Dill [1995]. Then the exact two-level hazard-free logic minimization problem is the complete cover problem in the cluster-cover framework. For the example in Figure 5(a), the ground set $E = \{e_1, e_2, e_3, e_4, e_5\}$ is illustrated in Figure 5(b). It can be easily verified that no pair of ground elements can be contained in a cube without intersecting the off-set. Therefore, the set of clusters is $\{\{e_1\}, \{e_2\}, \{e_3\}, \{e_4\}, \{e_5\}\}$. This is the cover—a unique solution to the complete cover problem. To complete the solution to the original logic minimization problem, we need to construct the cubes from the clusters. Consider cluster $\{e_1\}$. There are two maximal cubes that contain e_1: $\bar{c}\bar{d}$ and $\bar{a}\bar{d}$. Since cube $\bar{a}\bar{d}$ intersects with the 1–0 transition cube $t4$ without including its 1-end point, it is not a valid cube. Using such reasoning, we verify that the maximal valid cube that contains e_4 is e_4 itself. However, there are two valid maximal cubes that contain e_2, as illustrated in Figure 5(d). Note that the standard cost function assumes that each cube has cost 1; the last step here is meaningful only if we include literal-count as a secondary cost.

Nowick and Dill modified the Quine–McCluskey procedure into a constrained version to solve the exact two-level hazard-free logic minimization problem. We have shown that the problem can be solved in a more abstract framework. Our formalization brings some new insights. First, we define clusters directly from ground elements, whereas Nowick and Dill first generate the set of all prime implicants, and convert it into the set of dhf-prime implicants. It is well known that generating all prime implicants is a computationally involved task [Coudert and Madre 1992]. Second, we do not distinguish cubes that correspond to the same cluster; they are all equivalent as far as the optimization problem is concerned. Therefore, the size of the covering problem in our formulation is smaller than that of Nowick and Dill.

8.2 Multilayer Topological Planar Routing

We illustrate the problem of multi-layer topological planar routing [Cong et al. 1993] using an example in Figure 6, which has two rows of terminals marked by numbers. Terminals with the same number form a *net*, and each net is a ground element. A cluster is a set of nets that can be routed in a plane without crossing. For example, nets 1, 2, 3, and 4 in Figure 6 form a cluster. Nets 1 and 2 form another cluster. The maximum cluster problem

is to find a maximum set of nets that can be routed in one plane. The minimum cover problem is to find the minimum number of planes such that all the nets can be routed without crossing. In Figure 6, the maximal clusters are {1, 2, 3, 4}, {1, 2, 5}, and {3, 4, 6}. The minimum-layer routing is {{1, 2, 5}, {3, 4, 6}} and thus needs two layers, whereas the greedy peeling solution is {{1, 2, 3, 4}, {5,}, {6}} and requires three layers.

The formulation of multilayer topological routing in the framework of cluster-cover provides an efficient approach to finding exact solutions, taking advantage of techniques exploited originally in logic minimization. In fact, the prime covering technique has been implemented recently using ZBDDs for solving this problem [Coudert and Shi 1996a]. For all the routing examples solved previously by greedy peeling [Cong et al. 1993], exact optimum solutions are obtained in less than several CPU minutes. Furthermore, it has been observed [Coudert and Shi 1996a] that for all these examples, greedy solutions are surprisingly close to optimum solutions—typically less than 10% away. This is much stronger than the theoretical prediction: partial covering has a 63% ratio (Theorem 2), and complete covering has a logarithmic ratio (Theorem 3).

8.3 Dichotomy-Based Constrained Encoding

Constrained encoding is a fundamental problem that arises in race-free state assignment for asynchronous sequential machines, in delay-free realizations of asynchronous machines without essential hazards, in optimum state assignment for synchronous machines, and in programmable logic array decomposition [Shi and Brzozowski 1993; Unger 1969; Yang and Ciesielski 1991]. Here the so-called *dichotomy* is a pair of disjoint subsets of a given set of "states." For example, $a = (\{s_1, s_2\}, \{s_3\})$, $b = (\{s_2\}, \{s_4\})$, and $c = (\{s_1\}, \{s_2, s_3\})$ are dichotomies on the set $S = \{s_1, \ldots, s_4\}$. A bit assignment is a bipartition of S. For example, $x = (\{s_1, s_2\}, \{s_3, s_4\})$. A dichotomy is said to be satisfied by a bit assignment if no two states appearing in a single subset of the dichotomy appear in different subsets of the bit assignment. For example, dichotomies a and b are satisfied by x, but c is not. The constrained encoding problem is to find a minimum number of bit assignments that together satisfy all the dichotomies (exact satisfaction) or to find k bit assignments that satisfy as many dichotomies as possible (partial satisfaction).

Modeling of dichotomy-based constrained encoding illustrates some interesting aspects of the cluster-cover framework. Suppose that we are given two disjoint dichotomies $a = \{\{1\}, \{2\}\}$ and $b = \{\{3\}, \{4\}\}$, and dichotomy $c = \{\{1\}, \{4\}\}$. There are two dichotomies corresponding to $\{a, b\}$: $\{\{1, 3\}, \{2, 4\}\}$ and $\{\{1, 4\}, \{2, 3\}\}$. They are different in the sense that the first one and c can be satisfied by a bit assignment, whereas the second one and c cannot. This implies that we would have to have duplicate subsets $\{a, b\}$ (corresponding to two different dichotomies) to be included in the set of clusters; this violates our definition. This problem can be remedied by using ordered dichotomies. For each unordered dichotomy $\{p, q\}$, where p and q are two

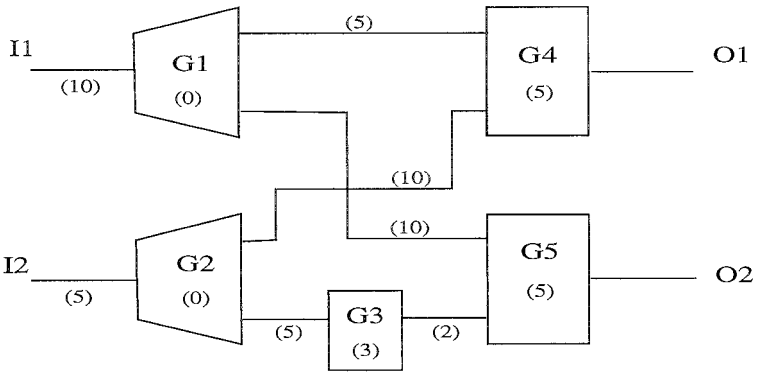

Fig. 7. Example of application timing assignment.

subsets of states, we replace it by two ordered dichotomies $\{p, q\}$ and $\{q, p\}$. Two ordered dichotomies are compatible if no state appears in the left block of one dichotomy and the right block of the other. We thus define the ground set to be the set of ordered dichotomies, and a cluster to be a set of ordered dichotomies that are compatible. This compatibility relation is transitive. Furthermore, the compatibility uniqueness is satisfied; therefore the maximum clique formulation stated in Section 3.4 can be used. However, only one of the two ordered dichotomies $\{p, q\}$ and $\{q, p\}$ needs to be covered in the final solution. This can be handled in the covering problem by renaming the two ground elements $\{p, q\}$ and $\{q, p\}$ as the same element. Note that changing unordered dichotomies into ordered ones is a commonly used method [Saldanha et al. 1991; Unger 1969].

With the preceding formulation, prime covering using ZBDDs has been implemented recently to solve the exact dichotomy-based constrained encoding problem; experimental results compare favorably with other state-of-the-art solvers [Coudert and Shi 1996b]. The application of greedy peeling has been described in Shi and Brzozowski [1993], which demonstrated that greedy solutions are near optimum, and optimum solutions can be achieved by several iterations of greedy peeling.

8.4 Application Timing Assignment for Delay-Fault Testing

The combinational circuit of Figure 7 has two primary inputs (I1 and I2), two primary outputs (O1 and O2), and five gates (G1 to G5) [Iyengar and Vijayan 1992]. Associated with each gate and wire is a *delay*: for example, G1 has delay 0, and the wire from I1 to G1 has delay 10. Delay-fault testing involves a sequence of test patterns at the primary inputs. Let T_i be the time of applying a signal to input i. For example, we may have the *application timing assignment* $T_{I1} = 0$ and $T_{I2} = 5$. For a timing assignment we denote by T_j the latest time for signals to arrive at output j from all the inputs. ($T_{O1} = \max(0 + 10 + 0 + 5 + 5, 5 + 5 + 0 + 10 + 5) = 25$, and $T_{O2} = \max(0 + 10 + 0 + 10 + 5, 5 + 5 + 0 + 5 + 3 + 2 + 5) = 25$.) For a path from input i to output j, the time allowed for signal propagation is

$T_j - T_i$, which may differ from the total delay of that path; this difference is the *slack* of the path. (Path I1 → G1 → G4 → O1 has $T_{I1} = 0$ and $T_{O1} = 25$. Since its delay is only 20, it has slack 5.) We define the *delay slack* for each delay as the minimum of the path slack of all input-to-output paths that include this delay. (The wire delay between G1 and G4 has slack 5.)

For timing assignment $T_{I1} = 0$ and $T_{I2} = 5$, only the wire delay between G1 and G4 has nonzero slack (5), and the sum of all the delay slack is 5. For timing assignment $T_{I1} = T_{I2} = 0$, we have $T_{O1} = 20$, and $T_{O2} = 25$. Now path I2 → G2 → G3 → G5 → O2 has slack 5. Also the wire delays between G2 and G3, and G3 and G5, have nonzero slack (5). The sum of the delay slack is 10.

A key observation in Iyengar and Vijayan [1992] is that delay slack affects the size of the delay fault detectable by any test set. To improve the quality of delay-fault testing, we need an application timing assignment that keeps delay slack as small as possible. However, the slack of a delay D cannot be made arbitrarily small. Let PI and PO be sets of primary inputs and primary outputs, respectively. Consider a primary input u and a primary output v. Let $d(u, v)$ be the length of a longest path from u to v minus the length of a longest path from u to v that includes delay D. Then, the *slack lower bound* for D is

$$\min\{d(u, v) | \forall (u, v),\ u \in PI \text{ and } v \in PO\}.$$

The *multiple-test application timing assignment* problem is as follows[3]: Given a fraction, α, $0 < \alpha \leq 1$, find the smallest number of application timing assignments such that, for α of the delays, there exists at least one timing assignment for each delay that enables it to achieve its slack lower bound. The *single-test application timing assignment* problem is to find an assignment such that as many delays as possible achieve their slack lower bounds.

In our framework, we take the circuit delays as ground elements. Then a subset of delays forms a cluster if all the delays in the subset can achieve their slack lower bounds under a single timing assignment; this is called a *consistent-tight set* in Iyengar and Vijayan [1992]. The multiple-test application timing assignment problem is the maximum k-cluster problem, whereas the single-test application timing assignment problem is the minimum α-cover problem.

The following two open questions are posed in Iyengar and Vijayan [1992].

> *"Can the TAT_multiple heuristic be improved to bring the number of assignments for $\beta = 1$ closer to that for $\beta = 1.1$? Is there a good lower bound for the number of assignments necessary to achieve $\beta = 1$?"*

For the first question, from Theorem 3 and the observation that greedy peeling is likely the best-possible approximation algorithm, it is very

[3]The formulation in Iyengar and Vijayan [1992] is slightly different.

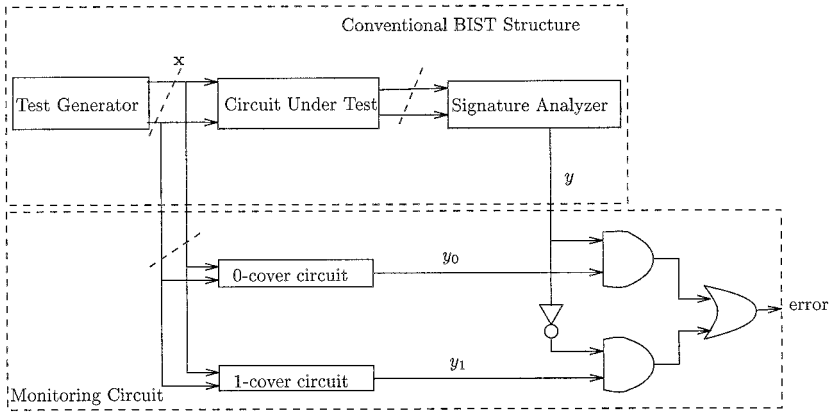

Fig. 8. An augmented BIST structure.

unlikely that the heuristic can be improved to bring the number of assignments for $\beta = 1$ closer to that for $\beta = 1.1$ in such a way that the heuristic still runs in polynomial time. For the second question, although we do not know a simple good lower bound yet, we have a tool, called prime covering, that can be used to find the number of assignments necessary to achieve $\beta = 1$.

8.5 Monitoring Logic Minimization for BIST Enhancement

Built-in self-test is widely used in VLSI. A typical BIST structure is shown in the upper part of Figure 8, which consists of a test generator, a combinational circuit under test (CUT), and a multiple-input signature analyzer. The test generator produces a test input sequence for the CUT. The signature analyzer then compresses the output sequence of the CUT into a signature, and compares it to the correct signature. If the two signatures differ, the CUT is faulty. However, if the CUT is faulty, the signature may be correct because of the data compression. This is called *aliasing*.

To overcome aliasing, Gössel and Jürgensen [1993] propose an improved BIST structure as shown in Figure 8. The output y is the first bit of the signature; we call this bit the *key*. Under the assumption that the CUT is correct, the value of the key for a given $\mathbf{x}$ is $f_0(\mathbf{x})$. The test sequence is assumed to include all the input combinations; hence f_0 is a completely specified function. It is assumed that in the presence of a fault, the output y can be specified by some other completely specified Boolean function f_i, $i = 1, \ldots, l$. For example, Table I shows three faults f_1, f_2, and f_3 of the correct function f_0 untestable because of aliasing.

For an input combination $\mathbf{x}$, we can distinguish the output y of the CUT into one of three cases: $y(\mathbf{x}) = 1$ when $f_0(\mathbf{x}) = 0$ (0-failure), $y(\mathbf{x}) = 0$ when $f_0(\mathbf{x}) = 1$ (1-failure), or $y(\mathbf{x}) = f_0(\mathbf{x})$ (no-failure). In this section three terminologies, input pattern, test pattern, and minterm, are used interchangeably. Given f_0 and f_i, $i = 1, \ldots, l$, as specified in Table I, we can

Table I. A Simple Example to Illustrate BIST Enhancement

test pattern x	f_0	f_1	f_2	f_3	y_0	y_1
000	0	0	0	1	1	0
100	0	0	0	0	–	0
010	0	0	0	0	–	0
110	1	0	1	1	0	–
001	0	1	1	0	1	0
101	1	1	1	1	0	–
011	1	0	1	1	0	–
111	1	1	1	1	0	–

partition the set of input patterns into three sets: 0-failure set P (i.e., if $\mathbf{x} \in P$, then there exists an f_i such that $f_0(\mathbf{x}) = 0$ and $f_i(\mathbf{x}) = 1$), 1-failure set Q (i.e., if $\mathbf{x} \in Q$, there exists an f_i such that $f_0(\mathbf{x}) = 1$ and $f_i(\mathbf{x}) = 0$), and no-failure set R (i.e., if $\mathbf{x} \in R$, then for all f_i, $i = 1, \ldots, l$, $f_i(\mathbf{x}) = f_0(\mathbf{x})$). Clearly P is a subset of the off-set of f_0, and Q is a subset of the on-set of f_0. We can represent the relationship between P (Q) and the set of faults f_i compactly in Tables II and III, where X indicates $f_i(\mathbf{x}) \neq f_0(\mathbf{x})$.

The basic idea of the monitoring circuit, called the error-detection circuit, is to predict the correct value of the key with the aid of two incompletely specified Boolean functions y_0 and y_1, where y_0 is used to monitor 0-failures, and y_1 for monitoring 1-failures. The error is given by the expression $y_0(\mathbf{x})y(\mathbf{x}) + y_1(\mathbf{x})\overline{y(\mathbf{x})}$.

Following Brayton et al. [1984], we are interested in specifying y_0 and y_1 using the on-set, off-set, and don't-care set. First, we consider how to determine the off-sets for y_0 and y_1. The error expression must evaluate to 0 if there is no failure ($y(\mathbf{x}) = f_0(\mathbf{x})$). This implies that the off-set of y_0 is the on-set of f_0, and the off-set of y_1 is the off-set of f_0. In our example, the off-set of y_0 is {110,101,011,111}, and the off-set of y_1 is {000,100,010,001}.

Next, we consider how to decide the on-sets and don't-care sets for y_0 and y_1. To monitor a 0-failure under $\mathbf{x}$, $y_0(\mathbf{x})$ must be such that $y_0(\mathbf{x}) = 1$ implies $f_0(\mathbf{x}) = 0$. Then if $y(\mathbf{x}) = 1$, $y_0(\mathbf{x})y(\mathbf{x}) = 1$, the error detection circuit evaluates to 1, and a 0-failure is detected. Similarly, $y_1(\mathbf{x})$ is chosen such that $y_1(\mathbf{x}) = 1$ implies $f_0(\mathbf{x}) = 1$. To monitor all the faults untestable by BIST due to aliasing, the on-set of $y_0(y_1)$ must be chosen from the 0-failure set P (1-failure set Q) so that, for each fault, there exists at least one 0-failure that is monitored by y_0 or at least one 1-failure that is monitored by y_1. The rest of the off-set (on-set) of f_0 after removing the chosen on-set of $y_0(y_1)$ forms the don't-care set of y_0 (y_1). In our example, we can choose {000,001} as the on-set for y_0. This will enable y_0 to monitor all three faults. Hence {100,010} will be the don't-care set for y_0. The on-set of y_1 can be chosen as empty, and {110,101,011,111} is the don't-care set of y_1. These results are summarized in the last two columns of Table I.

Table II. 0-Failure Set P

test pattern $\mathbf{x}$ in P	f_1	f_2	f_3
000			X
001	X	X	

Table III. 1-Failure Set Q

test pattern $\mathbf{x}$ in Q	f_1	f_2	f_3
110	X		
011	X		

In summary, the new logic minimization problem arising from the error-detection circuit design for BIST enhancement can be stated as follows. Given

(1) the off-set of y_0 (as the on-set of f_0),
(2) the off-set of y_1 (as the off-set of f_0),
(3) a set of faults $F = \{f_i \mid i = 1, \ldots, l\}$,
(4) the 0-failure set P, where each minterm detects a subset of faults from F (e.g., Table II), and
(5) the 1-failure set Q, where each minterm detects a subset of faults from F (e.g., Table III),

find a minimum-cost implementation of y_0 and y_1 such that the on-set of y_0 (y_1) is chosen from P (Q) and all the faults in F can be detected. This problem differs from the standard logic minimization problem where the on-set and the don't-care set are given explicitly. A two-step heuristic optimization procedure was presented in Gössel and Jürgensen [1993]. It consists of finding a minimum cardinality on-set (maximizing the cardinality of the don't-care set), and then solving a standard logic minimization problem. Unfortunately, both steps involve solving NP-complete problems.

In the following, we show that the problem can be solved nicely as one optimization problem using the cluster-cover framework. We note that each implicant of y_0 (y_1), called the y_0-implicant (y_1-implicant), is a cube that contains a set of minterms from P (Q) and does not contain any minterms from the off-set of y_0 (y_1). A cover is a set of implicants that detects all the faults. This structure is illustrated in Figure 9. Since the relationship between faults and minterms in P (Q) can be easily obtained from the given table (such as Table I) in linear time, the problem structure can be simplified so that it fits into our cluster-cover framework. The set of faults constitutes the ground set. A cluster is a set of faults that can be detected by those minterms that can be included in a single implicant. A cover is a set of clusters that includes all the faults. Now the minimization problem considered in Gössel and Jürgensen [1993] is to find a cover that contains as few clusters as possible.

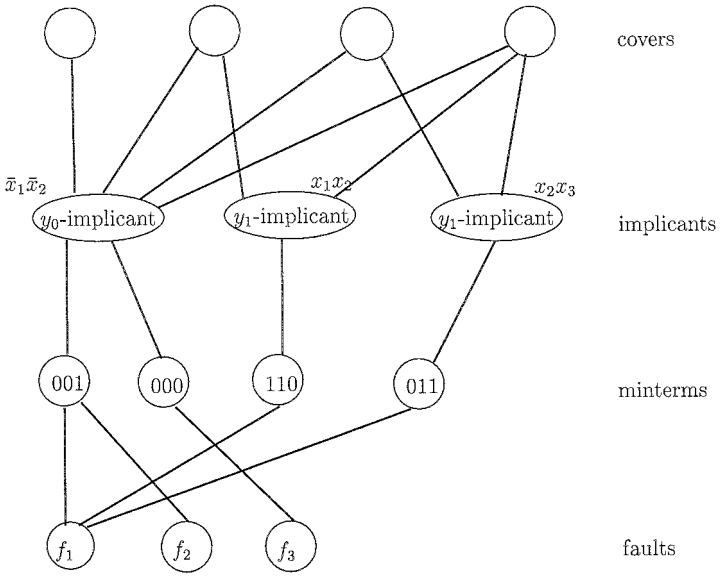

Fig. 9. Three-level hierarchy for monitoring-logic.

Our framework for the example is shown in Figure 10. Faults f_1, f_2, and f_3 form a y_0-cluster, because they can be detected under a set of minterms that can be included in the y_0-implicant $\bar{x}_1\bar{x}_2$. The minimum-cost cover is $\bar{x}_1\bar{x}_2$.

An interesting, and probably more practically relevant, extension of the work in Gössel and Jürgensen [1993] is the maximum k-cluster problem. Often the area that can be used for monitoring logic is limited. Then the problem is to design a monitoring circuit using this area so as to cover as many faults as possible.

Our cluster-cover framework leads to a precise formulation of the monitoring-logic minimization problem for BIST enhancement. Two paradigms described in this article provide, for the first time, both an exact algorithm and a complete heuristic for monitoring-logic minimization.

9. CONCLUSIONS

Systematic approaches to design and analyze algorithms for computationally difficult VLSI-CAD problems are highly desirable. We believe that this article has made one step towards this objective. We introduced a new mathematical notion called cluster-cover. We showed that it captures the combinatorial structure of a class of VLSI optimization problems, including two-level logic minimization, constrained encoding, multilayer topological planar routing, application timing assignment for delay-fault testing, and minimization of monitoring logic for BIST enhancement. These problems are apparently unrelated; however, we show in this article that they can be cast as two metaproblems on cluster covers.

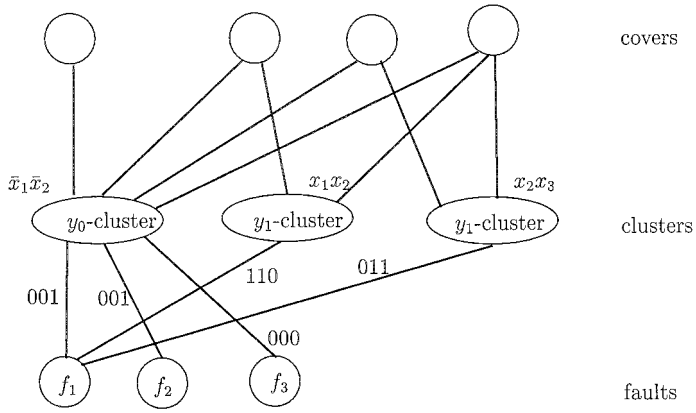

Fig. 10. A cluster-cover framework for the monitoring-logic example.

Two paradigms, greedy peeling and prime covering, are presented for developing heuristic and exact algorithms, respectively, for these two metaproblems. The paradigms capture generally applicable ingredients from previously developed algorithms for each individual application. This makes it possible to reuse established techniques in new problems, and provide new insights into existing problems.

Appendix A. Proof of Theorem 1

To prove the first part of the theorem, we consider the following binary relation $\mathcal{R}$ on E: For any $x \in E$ and $y \in E$, $x\mathcal{R}y$ if $\Pi(\{x, y\})$ is true, where Π is the compatibility predicate. From Condition (1) of subset-closed systems in the definition of cluster cover, $\mathcal{R}$ is reflexive. From the definition of $\mathcal{R}$, $\mathcal{R}$ is symmetric. Therefore, if $\mathcal{R}$ is also transitive, then $\mathcal{R}$ is an equivalence relation on E. Thus E is partitioned by the compatibility relation into a set of disjoint equivalence classes, each class being a cluster. Since MAX_h_CLUSTER is assumed to solve the maximum cluster problem exactly, greedy peeling yields an optimum solution for solving the maximum k-cluster problem.

For the second part of the theorem, note that any subset-closed system $(E, \mathcal{P})$ that obeys the combinatorial aspect of the Steinitz exchange principle is a matroid, as defined in Section 3.2. Matroid theory has established that, if $\mathcal{P}$ is explicitly given, the greedy algorithm always produces an optimal solution [Corman et al. 1990]. Although $\mathcal{P}$ is not explicitly given here, MAX_h_CLUSTER is assumed to solve the maximum cluster problem exactly. Thus, at each greedy step, the maximum cluster can always be obtained, and the theorem is proved. □

Appendix B. Proof of Theorem 2

We first prove the following lemma.

LEMMA 1. *Let F_k be a set of ground elements of maximal cardinality that could be covered by k clusters. Let $r = \lceil k/h \rceil$, and let E_i be the set of ground elements covered by almost-greedy peeling at the i*th *iteration, $1 \leq i \leq r$. Suppose that* MAX_h_CLUSTER *achieves a performance ratio ϵ; then*

$$|E_i| \geq \frac{\epsilon}{r}\left(|F_k| - \sum_{j=1}^{i-1} |E_j|\right).$$

PROOF. Suppose that $E_j \subset E$, $j < i$, is the set of ground elements covered by the jth iteration of almost-greedy peeling. Then, at the end of the $(i-1)$st iteration of almost-greedy peeling, the set of ground elements that still needs to be covered is:

$$E' = E - \cup_{j=1}^{i-1} E_j. \tag{1}$$

Our aim is to find out how the set of ground elements in E' that can be covered by the ith iteration of almost-greedy peeling relates to F_k. Without loss of generality, we can assume that $F_k = \cup_{a=1}^{r} E_a^*$, where $E_a^* \subseteq E$ is a subset of ground elements that can be covered by k clusters, and $E_a^* \cap E_b^* = \emptyset$ for any $a \neq b$, $a \leq r$, and $b \leq r$. Then, after the $i-1$ iterations of almost-greedy peeling, the remaining part of F_k in E' satisfies

$$F_k \cap E' = (\cup_{a=1}^{r} E_a^*) \cap E' = \cup_{a=1}^{r}(E_a^* \cap E'). \tag{2}$$

From (1), we have

$$F_k \cap E' = F_k \cap (E - \cup_{j=1}^{i-1} E_j). \tag{3}$$

Combining (2) and (3), we have the following equality:

$$\cup_{a=1}^{r}(E_a^* \cap E') = F_k \cap (E - \cup_{j=1}^{i-1} E_j).$$

So we have

$$|\cup_{a=1}^{r}(E_a^* \cap E')| = |F_k \cap (E - \cap_{j=1}^{i-1} E_j)|. \tag{4}$$

From our assumption, E_a^*, $a = 1, \ldots, r$ is a set of disjoint subsets of ground elements. Thus $E_a^* \cap E'$, $a = 1, \ldots, r$ are also disjoint. We can rewrite the left-hand side of (4) as follows.

$$|\cup_{a=1}^{r}(E_a^* \cap E')| = \sum_{a=1}^{r} |E_a^* \cap E'|.$$

Recall that E_j, $j < i$ is the set of ground elements covered by the jth iteration of almost-greedy peeling. Since E_j is to be peeled off from the ground set at the jth iteration of almost-greedy peeling, it follows that the

$E_j, j = 1, \ldots, i-1$, are disjoint. Thus we can rewrite the right-hand side of (4) as follows.

$$\begin{aligned}
|F_k \cap (E - \cup_{j=1}^{i-1} E_j)| &= |F_k \cap (E \cap \overline{\cup_{j=1}^{i-1} E_j})| \\
&= |(F_k \cap E) \cap \overline{\cup_{j=1}^{i-1} E_j}| \\
&= |F_k \cap \overline{\cup_{j=1}^{i-1} E_j}| \\
&= |F_k - \cup_{j=1}^{i-1} E_j| \\
&\geq |F_k| - \sum_{j=1}^{i-1} |E_j|.
\end{aligned}$$

In summary, (4) can be rewritten as:

$$\sum_{a=1}^{r} |E_a^* \cap E'| \geq |F_k| - \sum_{j=1}^{i-1} |E_j|.$$

By the pigeonhole principle, there must exist an x, $1 \leq x \leq r$ such that

$$|E_x^* \cap E'| \geq \frac{1}{r}\left(|F_k| - \sum_{j=1}^{i-1} |E_j|\right).$$

After $i-1$ iterations of almost-greedy peeling, the maximal possible h-cluster must be at least as large as $E_x^* \cap E'$. Because MAX_h_CLUSTER has performance ratio ϵ, the set chosen by the ith iteration of almost-greedy peeling satisfies

$$|E_i| \geq \epsilon |E_x^* \cap E'|.$$

Therefore

$$|E_i| \geq \frac{\epsilon}{r}\left(|F_k| - \sum_{j=1}^{i-1} |E_j|\right). \qquad \square$$

Now we are ready to prove Theorem 2. Let F_k be a set of ground elements of maximal cardinality that can be covered by k clusters. Let E_i be the set of ground elements covered by almost-greedy peeling at the ith iteration, $1 \leq i \leq r$. Then the performance ratio is

$$\gamma = \frac{\sum_{i=1}^{r} |E_i|}{|F_k|}. \qquad (5)$$

By Lemma 1, we have

$$|E_i| \geq \frac{\epsilon}{r}\left(|F_k| - \sum_{j=1}^{i-1} |E_j|\right).$$

Thus

$$\sum_{i=1}^{r} |E_i| \geq \frac{\epsilon}{r} \sum_{i=1}^{r}\left(|F_k| - \sum_{j=1}^{i-1} |E_j|\right). \tag{6}$$

First, let us define a new sequence, w_i, $i = 1, \ldots, r$, recursively as follows.

$$w_i = \begin{cases} \dfrac{\epsilon}{r}|F_k|, & i = 1 \\ \dfrac{\epsilon}{r}(|F_k| - \Sigma_{j=1}^{i-1} w_j), & i = 2, \ldots, r. \end{cases}$$

Note that

$$\begin{aligned} w_i &= \frac{\epsilon}{r}\left(|F_k| - \sum_{j=1}^{i-1} w_j\right) \\ &= \frac{\epsilon}{r}\left(|F_k| - \sum_{j=1}^{i-2} w_j\right) - \frac{\epsilon}{r} w_{i-1} \\ &= \left(1 - \frac{\epsilon}{r}\right) w_{i-1} \\ &= \left(1 - \frac{\epsilon}{r}\right)^{i-1} w_1. \end{aligned}$$

This means that w_i, $i = 1, \ldots, r$ form a geometric sequence. Let $q = 1 - (\epsilon/r)$. Since

$$\sum_{i=1}^{r} q^{i-1} = \frac{1 - q^r}{1 - q},$$

we have

$$\sum_{i=1}^{r} w_i = \sum_{i=1}^{r} (q^{i-1} w_1) = \left(\sum_{i=1}^{r} (q^{i-1}) \right) w_1 = \frac{1 - q^r}{1 - q} \frac{\epsilon}{r} |F_k|.$$

Simplifying the last expression yields

$$\sum_{i=1}^{r} w_i = \left(1 - \left(1 - \frac{\epsilon}{r} \right)^r \right) |F_k|. \tag{7}$$

Now, by comparing (7) with (5) and (6), we see that, in order to prove Theorem 2, we only need to show that

$$\sum_{i=1}^{r} |E_i| \geq \sum_{i=1}^{r} w_i,$$

because

$$\gamma = \frac{\Sigma_{i=1}^{r} |E_i|}{|F_k|}$$

and

$$1 - \left(1 - \frac{\epsilon}{r} \right)^r = \frac{\Sigma_{i=1}^{r} w_i}{|F_k|}.$$

This can be done by induction on r. Clearly, the preceding inequality holds for $r = 1$. We now assume that $r > 1$, and that the inequality holds for $r - 1$. Then we have

$$\sum_{i=1}^{r} |E_i| \geq \sum_{i=1}^{r-1} |E_i| + \frac{\epsilon}{r} \left(|F_k| - \sum_{i=1}^{r-1} |E_i| \right)$$

$$= \frac{\epsilon}{r} |F_k| + \left(1 - \frac{\epsilon}{r} \right) \sum_{i=1}^{r-1} |E_i|$$

$$\geq \frac{\epsilon}{r} |F_k| + \left(1 - \frac{\epsilon}{r} \right) \sum_{i=1}^{r-1} w_i$$

$$= \sum_{i=1}^{r-1} w_i + \frac{\epsilon}{r}\left(|F_k| - \sum_{i=1}^{r-1} w_i \right)$$

$$= \sum_{i=1}^{r} w_i .$$

Hence, the theorem is proved. □

Appendix C. Proof of Theorem 3

For $P_i \in \mathcal{P}$, each time a set, say Q, of elements is peeled off by greedy peeling, we remove the elements of Q from P_{ij}, that is, $P_i \leftarrow P_i - Q$. Each time a maximal cluster $P^{(k)}$ is picked and added to C, the algorithm incurs a cost of 1. We spread this cost evenly among the elements covered for the first time in $P^{(k)}$; that is, each element e in $P^{(k)}$ is assigned a cost of $c_e = (1/|P^{(k)}|)$. We have

$$\sum_{e \in E} c_e = |C|,$$

where C is the α-cover returned by GREEDY. Note that since α may not be equal to 1, for some e, c_e may be 0.

Let E' be the set of elements covered by C. Then

$$|E'| = |\cup_{P_i \in C} P_i| \geq \alpha |E|.$$

Let C' be an arbitrary α-cover that covers E'. Then

$$|\cup_{P_i \in C'} P_i| \geq \alpha |E|,$$

and

$$\sum_{P_i \in C'} W_i \geq \sum_{e \in E} c_e$$

$$= |C|.$$

Since an optimal cover, denoted by C^*, is one of these C', we have,

$$\sum_{P_i \in C^*} W_i \geq |C|.$$

Thus

$$|C| \leq |C^*| \frac{\Sigma_{P_i \in C^*} W_i}{|C^*|}.$$

We can rewrite this inequality as

$$\frac{|C|}{|C^*|} \leq \frac{\Sigma_{P_i \in C^*} W_i}{|C^*|}$$

$$= \text{average } W_i \text{ in } C^*$$

$$\leq \max\{W_i | P_i \in \mathcal{P}\}$$

$$= \mu.$$

To show that μ is bounded by $H(|P|_{\max})$, where $|P|_{\max}$ is the size of the first cluster picked by greedy peeling, we prove that for any $P_i \in \mathcal{P}$,

$$W_i \leq H(|P_i|).$$

Let us denote the size of P_i after $k - 1$ iterations by $|P_i^{(k-1)}|$. Then the cost added to W_i at iteration k (line 9 in GREEDY) is

$$\Delta(W_i)^{(k)} = (|P_i^{(k-1)}| - |P_i^{(k)}|)/|P^{(k)}|,$$

where $P^{(k)}$ is the largest cluster picked at iteration k. Since $|P^{(k)}| \geq |P_i^{(k-1)}|$ for any P_i, we have

$$\Delta(W_i)^{(k)} \leq (|P_i^{(k-1)}| - |P_i^{(k)}|)/|P_i^{(k-1)}|.$$

So the total cost is

$$W_i = \sum_{k=1}^{|C|} \Delta(W_i)^{(k)}$$

$$\leq \sum_{k=1}^{|C|} (|P_i^{(k-1)}| - |P_i^{(k)}|)/|P_i^{(k-1)}|.$$

For integers a and b, where $a < b$, we have

$$H(b) - H(a) = \sum_{i=a+1}^{b} \frac{1}{i}$$

$$\geq (b - a)\,\frac{1}{b}.$$

Using this inequality, we obtain the telescoping sum

$$\begin{aligned} W_i &\leq \sum_{k=1}^{|C|} (H(|P_i^{(k-1)}|) - H(|P_i^{(k)}|)) \\ &= H(|P_i^{(0)}|) - H(|P_i^{(|C|)}|) \\ &= H(|P_i^{(0)}|) - H(0) \\ &= H(|P_i^{(0)}|) \\ &= H(|P_i|), \end{aligned}$$

since $|P^{|C|}| = 0$ and $H(0) = 0$. This completes the proof of Theorem 3. □

REFERENCES

BRAYTON, R. K., HACHTEL, G. D., MCMULLEN, C. T., AND SANGIOVANNI-VINCENTELLI, A. L. 1984. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic, New York.

CHVATAL, V. 1979. A greedy heuristic for the set covering problem. *Math. Oper. Res. 4,* 233–235.

CONG, J., HOSSAIN, M., AND SHERWANI, N. A. 1993. A provably good multilayer topological planar routing algorithm in IC layout designs. *IEEE Trans. Comput. Aided Des. 12,* 1 (Jan.), 70–78.

CORMAN, T. H., LEISERSON, C. E., AND RIVEST, R. L. 1990. *Introduction to Algorithms*. The MIT Press, Cambridge, MA.

COUDERT, O. AND MADRE, J. C. 1992. Implicit and incremental computation of primes and essential primes of Boolean functions. In *ACM/IEEE Design Automation Conference* (Anaheim, CA, June 8–12, 1992), 36–39.

COUDERT, O. AND MADRE, J. C. 1995. New ideas for solving covering problems. In *ACM/IEEE Design Automation Conference* (San Francisco, CA, June 12–16, 1995), 641–646.

COUDERT, O. AND SHI, C.-J. 1996a. Exact multi-layer topological planar routing. In *Proceedings of the IEEE Custom Integrated Circuit Conference (CICC'96)* (San Diego, CA, May 5–8), 179–182.

COUDERT, O. AND SHI, C.-J. 1996b. Exact dichotomy-based constrained encoding. In *Proceedings of the IEEE International Conference on Computer Design (ICCD'96)* (Austin, TX, Oct.), 426–431.

FAIGLE, U. 1979. The greedy algorithm for partially ordered sets. *Discrete Math. 28,* 153–159.

GÖSSEL, M. AND JÜRGENSEN, H. 1993. Monitoring BIST by covers. In *Proceedings of EuroDAC'93—European Design Automation Conference* (Hamburg, Germany, Sept.) 208–213.

IYENGAR, V. S. AND VIJAYAN, G. 1992. Optimized test application timing for AC test. *IEEE Trans. Comput. Aided Des. 11,* 11 (Nov.), 1439–1449.

JOHNSON, D. S. 1974. Approximating algorithms for combinatorial problems. *J. Comput. Syst. Sci. 9,* 256–278.

KOHAVI, Z. 1978. *Switching and Finite Automata Theory*, second edition. McGraw-Hill, New York.

KORTE, B. AND LOVÁSZ, L. 1984. Greedoids—A structural framework for the greedy algorithm. In *Progress in Combinatorial Optimization*, W. Pulleyblank, Ed., Academic Press, New York, NY, 221–243.

LIN, B., COUDERT, O., AND MADRE, J. C. 1992. Symbolic prime generation for multiple-valued functions. In *ACM/IEEE Design Automation Conference* (Anaheim, CA, June 8–12, 1992), 40–44.

LOVÁSZ, L. 1975. On the ratio of optimal integral and fractional covers. *Discrete Math. 13,* 383–390.

LUND, C. AND YANNAKAKIS, M. 1993. On the hardness of approximating minimization problems. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing* (San Diego, CA, May 16–18, 1993), 286–293.

MCGEER, P., SANGHAVI, J., AND BRAYTON, R. 1993. Espresso-signature: A new exact minimizer for logic functions. In *ACM/IEEE Design Automation Conference* (Dallas, TX, June 14–18, 1993), 618–624.

MINATO, S. 1993. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *ACM/IEEE Design Automation Conference* (Dallas, TX, June 14–18, 1993), 272–277.

NOWICK, S. M. AND DILL, D. D. 1995. Exact two-level minimization of hazard-free logic with multiple-input changes. *IEEE Trans. Comput. Aided Des. 14,* 8 (Sept.), 986–997.

RUDELL, R. L. AND SANGIOVANNI-VINCENTELLI, A. L. 1987. Multiple-valued minimization for PLA optimization. *IEEE Trans. Comput. Aided Des. 6,* 5 (Sept.), 727–750.

SALDANHA, A., VILLA, T., BRAYTON, R. K., AND SANGIOVANNI-VINCENTELLI, A. L. 1991. A framework for satisfying input and output encoding constraints. In *Proceedings of the 28th IEEE/ACM Design Automat. Conference* (San Francisco, CA, June 17–21, 1991), 170–175.

SHI, C.-J. 1993. Optimum logic encoding and layout wiring for VLSI design: A graph-theoretic approach. Ph.D. Thesis, Dept. of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, Dec. (Tech. Rep. CS-93-60).

SHI, C.-J. 1997. Block-level fault isolation using partition theory and logic minimization techniques. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC'97)* (Chiba, Japan, Jan.), 319–324.

SHI, C.-J. AND BRZOZOWSKI, J. A. 1993. An efficient algorithm for constrained encoding and its applications. *IEEE Trans. Comput. Aided Des. 13,* 12 (Dec.), 1813–1826.

SHI, C.-J. AND BRZOZOWSKI, J. A. 1995. A framework for the analysis and design of algorithms for a class of VLSI-CAD optimization problems. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC'95)* (Chiba, Japan, Aug.) 67–74.

TRACEY, J. H. 1966. Internal state assignment for asynchronous sequential machines. *IEEE Trans. Electron. Comput.* (Aug.), 551–560.

UNGER, S. H. 1969. *Asynchronous Sequential Switching Circuits*, John Wiley & Sons, New York.

YANG, S. AND CIESIELSKI, M. J. 1991. Optimum and suboptimum algorithms for input encoding and its relationship to logic minimization. *IEEE Trans. Comput. Aided Des. 10,* 1 (Jan.), 4–12.

Received January 1996; revised February 1997; accepted May 1997