On Computing the Minimum Feedback Vertex Set of a Directed Graph by Contraction Operations

Hen-Ming Lin and Jing-Yang Jou

***Abstract*—Finding the minimum feedback vertex set (MFVS) in a graph is an important problem for a variety of computer-aided design (CAD) applications and graph reduction plays an important role in solving this intractable problem. This paper is largely concerned with three new and powerful reduction operations. Each of these operations defines a new class of graphs, strictly larger than the class of contractible graphs [Levy and Low (1988)], in which the MFVS can be found in polynomial-time complexity. Based on these operations, an exact algorithm run on branch and bound manner is developed. This exact algorithm uses a good heuristic to find out an initial solution and a good bounding strategy to prune the solution space. To demonstrate the efficiency of our algorithms, we have implemented our algorithms and applied them to solving the partial scan problem in ISCAS'89 benchmarks. The experimental results show that if our three new contraction operations are applied, 27 out of 31 circuits in ISCAS'89 benchmarks can be fully reduced. Otherwise, only 12 out of 31 can be fully reduced. Furthermore, for all ISCAS'89 benchmarks our exact algorithm can find the exact cutsets in less than 3 s (CPU time) on SUN-UltraII workstation. Therefore, the new contraction operations and our algorithms are demonstrated to be very effective in the partial scan application.**

***Index Terms*—Algorithm, branch and bound, design for testability, graph theory, minimum feedback vertex set, partial scan.**

I. Introduction

GIVEN a directed graph $G = (V, E)$, the *minimum feedback vertex set (MFVS)* problem is to identify the minimum subset V' in V such that after removing all vertices in V', the remaining graph is acyclic. Many computer-adided design (CAD) applications can be reduced into this problem. The following three applications are pointed out in [2]. 1) Identify the state variables needed in test generation. 2) Identify the feedback cut needed in the logic simulation. 3) Identify the location in a circuit needed for complete diagnosis of circuit faults. Xie and Beerel [3] transform an original *Markov chain*, which models an asynchronous system under analysis, into a smaller one. The *feedback vertex set* (FVS) of the state transition graph of the original Markov chain is the state space of the transformed Markov chain. Because larger state space of the Markov chain can slow down the analysis significantly, the size of the FVS plays an important role in speeding up the analysis. *Partial scan problem* is another application on the MFVS problem. It only makes a subset of flip-flops scannable [4] and thus reduces both the area overhead and the performance degradation caused by the scan design. In the past, the authors of [5]–[11] used a directed graph, called *S-graph*, to model the structural dependencies in a circuit. In an S-graph $G = (V, E)$, a vertex i represents a flip-flop i in the original circuit and an edge (i, j) exists when there is a combinational path from the flip-flop i to the flip-flop j. Then, they used the MFVS of the S-graph to intelligently choose the flip-flops to scan.

Unfortunately, the MFVS problem is known as NP-complete on general graph [12] and even on bipartite one [13]. Smith and Walford [2] develop an optimal algorithm to solve this problem in any graph. Their major idea is to propose a sufficient condition for one or more vertices to be a part of some MFVS. This sufficient condition is the following: given a directed graph $G = (V, E)$, a vertex set $F \subseteq V$ and a vertex set G_F that consists of all vertices in the cycles containing a vertex in F, F is a part of some MFVS of G if F is an MFVS of G_F. Their algorithm chooses one vertex at a time, two vertices at a time, and so on as the F set and tests if the chosen vertex set satisfies this condition. If a vertex set F satisfies the condition, the vertex set F is removed from the graph. The process is then repeated on the remaining graph. However, in the worst case, e.g., *complete graph*, this algorithm can only find the MFVS after enumerating the exponential number of all vertex combinations. Therefore, their algorithm can only handle smaller graphs [10]. Ashar and Malik [9] propose an implicit method based on binary decision diagram (BDD) to compute the MFVS in any graph. They derive a Boolean function whose satisfying assignments directly correspond to the FVS of a graph. By searching the shortest path in the BDD of this Boolean function, the MFVS of the graph can be found. Chakradhar *et al.* [7] use the *integer linear programming* (ILP) to model the MFVS problem. A lower bound can be calculated by relaxing their ILP formulation to *linear programming* (LP) formulation. They then use this lower bound to design a branch and bound algorithm to solve the MFVS problem.

Because of the intractability of the MFVS problem, some researches focus on designing good approximation algorithms. These algorithms are designed with theoretical supports so that the ratio of their result to the optimal result has an upper bound called *approximation factor*. Leighton and Rao [14] have proven that for any n-node *multicommodity flow problem with uniform demands*, the *max-flow* is within a $O(\log n)$-factor of the *min-cut*, i.e., the max-flow is smaller than $O(\log n)$ times the min-cut. According to this theorem, they first design

Manuscript received June 28, 1999; revised September 30, 1999. This work was supported in part by Republic of China National Science Council under Grant NSC89-2215-E-009-058. This paper was recommended by Associate Editro R. Gupta.

H.-M. Lin is with the Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan 300, R.O.C. (e-mail: hermes@eda.ee.nctu.edu.tw).

J.-Y. Jou is with the Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan 300, R.O.C. (e-mail: jyjou@bestmap.ee.nctu.edu.tw).

Publisher Item Identifier S 0278-0070(00)02743-3.

0278–0070/00$10.00 © 2000 IEEE

an approximation algorithm with the approximation factor $O(\log n)$ for finding a *directed cut* that splits the graph into two (approximately) equal-sized components. Then, they design an approximation algorithm for MFVS problem by using the directed cut. This approximation algorithm for MFVS problem partitions the original graph into two parts by a directed cut, adds the cut into the feedback set, and continues finding the directed cut for each part recursively. The recursion ends when the graph no longer contains cycles. The approximation factor of this algorithm is $O(\log^2 n)$. Even *et al.* [15] further improve the approximation factor to $O(\log n \log\log n)$ by changing the method of selecting the directed cut. They first introduce a distance metric on the vertex/edge set. Then, their approximation algorithm picks arbitrarily a vertex s and conducts a single-source-shortest-paths algorithm from s. The single-source-shortest-paths algorithm defines layers with respect to the source s, where each layer is a directed cut, and the layer with the minimum cost is chosen to partition the graph into two parts. Bar-Yehuda *et al.* [16] also propose an approximation algorithm according to the property of *2–3 subgraph* in which the degree of each vertex is either two or three. The approximation factor of their algorithm is $O(4 - 2/n)$. However, this algorithm can only handle undirected graphs.

Even though MFVS problem is NP-complete, the MFVS of some classes of graphs can be found in polynomial-time complexity. Several authors have proposed polynomial-time algorithms to solve the MFVS problem in some special graphs. Liang [19] proposes an algorithm with $O(|E||V|)$ time complexity to solve the *minimum cost feedback vertex set* (MCFVS) problem in *permutation graphs*. Lu and Tang [18] propose a linear time algorithm to find the MCFVS of *interval graphs*. However, most of the graphs modeling the CAD applications do not generally satisfy the restrictions caused by these special graphs. Levy and Low [1] propose five contraction operations. The five operations define a class of graphs called *contractible graphs* in which the MFVS can be found in polynomial-time complexity. Their algorithm reduces the size of an input graph by iteratively using the five contraction operations. If a graph can be fully reduced, the graph is a contractible graph. Levy and Low [1] have shown that the class of contractible graphs contains both the class of *cyclically reducible graphs* [17] and the class of interval graphs. In this paper, we further propose three new and powerful contraction operations. Incorporating our new contraction operations with the five operations proposed in [1], we define a new class of graphs called *reducible graphs*, strictly larger than the class of contractible graphs, in which the MFVS can be found in polynomial-time complexity.

Please note that the three new contraction operations can enhance both the speed of the existent exact algorithms [7], [9] and the quality of the FVS produced by the existent approximation algorithms [14], [15]. The reason is that both the speed of the exact algorithms [7], [9] and the approximation factors of the approximation algorithms [14], [15] depend on the size of input graph, and the graph reduction caused by the eightcontraction operations (three new plus five old) can reduce the size of the input graph while it still preserves the information necessary for finding a MFVS. The property is called *optimality preserving property*. Therefore, even for the graph not belonging to the class of reducible graphs, the contraction operations could still reduce their size and improve the performance of these existent algorithms. Ashar, Malik [9] and Chakradhar *et al.* [7] have shown that the graph reduction can significantly speed up their exact algorithms.

Furthermore, we develop a new exact algorithm according to the contraction operations. This algorithm runs on the branch and bound manner and thus needs an effective bounding strategy and a good initial solution. Because of these requirements, we develop an effective bounding algorithm to prune the solution space and a heuristic algorithm that produces a good initial solution. Both the algorithms are based on the contraction operations and thus reduce the effort of implementation. To demonstrate the efficiency of our algorithms, we have implemented our algorithms and applied them to the partial scan application. The experimental results show that if our three new contraction operations are applied, 27 out of 31 circuits in ISCAS'89 benchmarks [4], [8], [10], [11] can be fully reduced. Otherwise, only 12 out of 31 can be fully reduced. Furthermore, for all ISCAS'89 benchmarks our exact algorithm can find out the exact cutsets in less than 3 s (CPU time) on SUN-UltraII workstation. Therefore, the new contraction operations and our algorithms are demonstrated to be very effective in the partial scan application.

The remainder of this paper is organized as follows. We state preliminaries in Section II. In Section III and Section IV, we propose the new contraction operations and our exact algorithm. The experimental results present in Section V. Finally, Section VI concludes the paper.

II. Preliminaries

First, some definitions are given and will be used in the following discussions.

Definition 1: A *directed graph* G is a pair (V, E), where V is a finite set and E is a binary relation on V. The set V is called the *vertex set* of G, and its elements are called *vertices*. The set E is called the *edge set* of G, and its elements are called *edges*. The *size* of a graph $G = (V, E)$ is the number of the vertices in the vertex set V.

Definition 2: If (u, v) is an edge in a directed graph $G = (V, E)$, we say that it *starts* at the vertex u and *points* to the vertex v, and it is an *out-edge* of the vertex u and an *in-edge* of the vertex v. The *out-degree* of the vertex u is the number of the out-edges of u, and the *in-degree* of the vertex u is the number of the in-edges of u. The *degree* of a vertex is its in-degree plus its out-degree.

Definition 3: A *path* of length k from a vertex u to a vertex u' in a graph $G = (V, E)$ is a sequence $\langle v_0, v_1, v_2, \ldots, v_k \rangle$ of vertices such that $u = v_0$, $u' = v_k$, and $(v_{i-1}, v_i) \in E$ for $i = 1, 2, \ldots, k$. If there is a path p from a vertex u to another vertex u', we say that u' is *reachable* from u via p. A path forms a *cycle* if $v_0 = v_k$ and the path contains at least one edge. A cycle $\langle v_0, v_0 \rangle$ is called a *self-loop*.

Definition 4: A *cutset* of the graph $G = (V, E)$ is a set of vertices $C \subseteq V$ such that G–C is acyclic.

Now, we formally state the MFVS problem.

The MFVS Problem (Cycle Breaking Problem): Given a directed graph G, find a cutset C such that the number of the vertices in the cutset C is the minimum.

In Section II-A and II-B, we discuss the previous contraction algorithms that are used frequently to solve the MFVS problem [5]–[11] and their applications on the partial scan problems.

A. The Previous Contraction Algorithms

The previous contraction algorithms proposed in [1] and [10] consist of two major steps: 1) the graph reduction and 2) the heuristic selection. The graph reduction step reduces the graph while it preserves all properties necessary for finding the minimum cutset. In the graph reduction step, there are five contraction operations: 1) IN0 operation: given a vertex v with in-degree zero, remove v and all its incident edges; 2) OUT0 operation: given a vertex v with out-degree zero, remove v and all its incident edges; 3) LOOP operation: given a vertex v with a self-loop, push v into the cutset, remove v and all its incident edges; 4) IN1 operation: given a vertex v with in-degree one and no self-loop, u is the unique *predecessor* of v, merge v into u as a single vertex; and 5) OUT1 operation: given a vertex v with out-degree one and no self-loop, u is the unique *successor* of v, merge v to u as a single vertex. Fig. 1 shows an example of applying the contraction operations to a directed graph. If the graph can be reduced completely by the contraction operations, the cutset is the minimum one, and this graph belongs to the *contractible graph*. Otherwise, a vertex is selected heuristically from the remaining vertices, push it into the cutset and remove it and its incident edges from the graph. In [1], the heuristic is conducted by randomly choosing a vertex from the remaining graph. From several experiments, the authors of [10] found that random selection is not effective. They proposed to select the vertex with the maximum sum of the in-degree and the out-degree in the remaining graph. If the heuristic selection is conducted, then the cutset obtained is not guaranteed to be the minimum. The graph reduction step and the heuristic selection step are iteratively applied until the graph is empty.

B. Partial Scan

The *scan design* is a popular *design-for-testability* (DFT) technique. In a *full scan design*, all flip-flops are replaced by scannable flip-flops, so that only combinational test generation method is necessary for generating the test set with high fault coverage. However, the full scan designs often incur high area overhead and performance degradation. Due to its low overhead, the *partial scan design* methodology has become a major DFT technique for sequential circuits. For a long time, a variety of techniques have been applied to automatically selecting the scan flip-flops. The approach proposed in [4] uses testability analysis to select scan flip-flops. The method suggested in [20] uses functional vectors and a combinational test generator to select the scan flip-flops. In [8], the approach is concerned with the selection of scan flip-flops aiming to break up the cyclic structure in the circuits. In this paper, we focus on this approach that selects the scan flip-flops according to the MFVS of *S-graph*. The *S-graph* models the structural dependencies between flip-flops by a directed graph. In an S-graph $G = (V, E)$, a vertex i represents a flip-flop i in the original circuit and an edge (i, j) exists when there is a combinational path from the flip-flop i to the flip-flop j. Primary inputs and primary outputs are not included in the vertex set V. Therefore, the problem of scan flip-flop selection is reduced to find the MFVS of S-graph. This approach tends to produce high fault coverage with relatively low area overhead. We will demonstrate the efficiency of our algorithms on this approach.

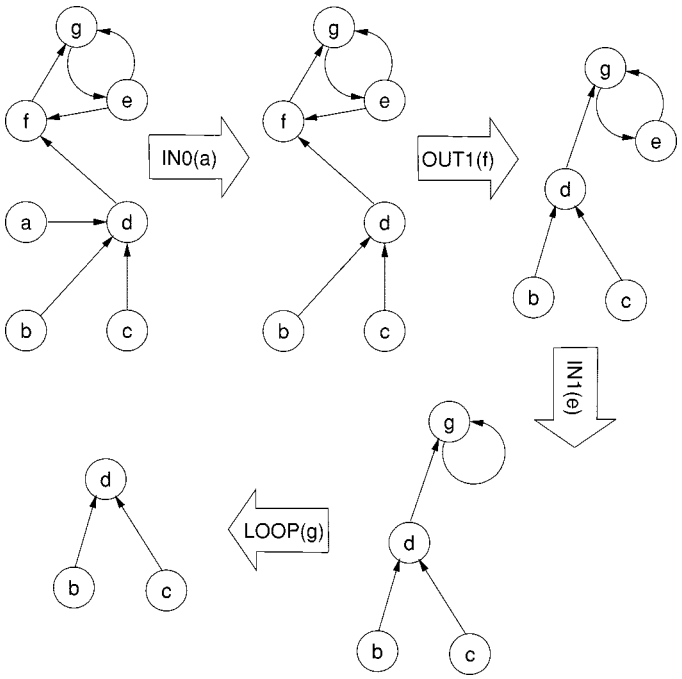

Fig. 1. Contraction operations on a directed graph.

III. Our New Contraction Operations

A. Π-edges (PIE) Operation

PIE operation removes the redundant edges of a graph G for cycle breaking consideration. It contains two ideas. We describe them as shown below.

The first idea comes from the concept of *acyclic edges*. Given an edge e, if no cycle goes through e, the edge e is called *acyclic edge*. For illustration in Fig. 2(a), there are two subgraphs, G_1 and G_2, in the graph G. The set of the edges between G_1 and G_2 is called $E_{G1,G2}$. If all edges in the $E_{G1,G2}$ are in the same direction, such as from G_1 to G_2, then no cycle in G contains the edges in the $E_{G1,G2}$. Therefore, the edges in the $E_{G1,G2}$ are acyclic edges and can be removed from G for cycle breaking consideration. The graph G shown in Fig. 2(b) is not the contractible one because both the in-degree and the out-degree of all vertices in G are larger than one and there is no self-loop. However, Fig. 2(b) can be divided into two subgraphs, G_1 and G_2 as shown, and the acyclic edges shown in bold edges can be removed. Because both G_1 and G_2 are the contractible graphs and can be reduced as shown in Fig. 2(c), the optimum solution $\{a, b, f, h\}$ can be found.

The following discussion explains how to identify the acyclic edges.

Definition 5: A *strongly connected component* of a directed graph $G = (V, E)$ is the maximum set of vertices $U \subseteq V$ such

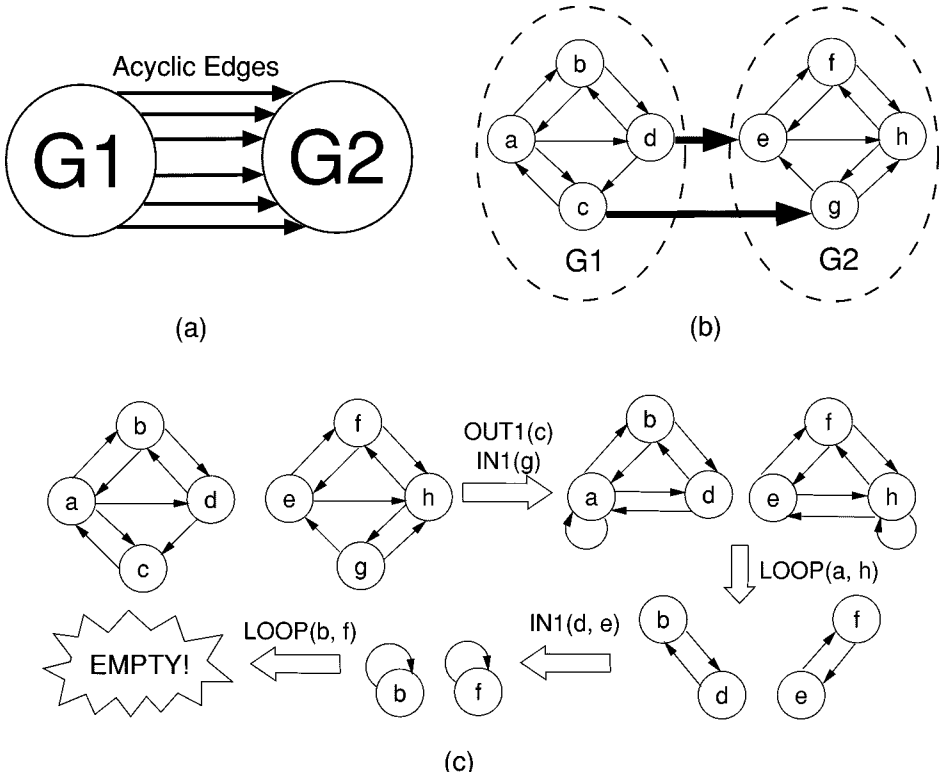

Fig. 2. (a) The concept of acyclic edges. (b) A noncontractible graph. (c) Graph reduction.

that for every pair of vertices u and v in U, the vertices u and v are reachable from each other [21].

Definition 6: The $G^{\mathrm{SCC}} = (V^{\mathrm{SCC}}, E^{\mathrm{SCC}})$ is the *component graph* of $G = (V, E)$ where V^{SCC} contains one vertex for each strongly connected component of G and E^{SCC} contains the edge (u, v) if there is a directed edge from a vertex in the strongly connected component of G corresponding to u to a vertex in the strongly connected component of G corresponding to v [21].

For the sake of the clarity, the proofs of the following lemmas, theorems, and corollaries are placed into the Appendix.

Theorem 1: G^{SCC} is a directed acyclic graph [21].

Corollary 1: Given a graph $G = (V, E)$ and its corresponding component graph $G^{\mathrm{SCC}} = (V^{\mathrm{SCC}}, E^{\mathrm{SCC}})$, the edges in E associated with an edge in E^{SCC} do not belong to any cycle in G and can be eliminated from G for the cycle breaking consideration.

According to Theorem 1 and Corollary 1, in order to find the acyclic edges between subgraphs, we should find the component graph of G first. Those edges associated with the edges in the component graph G^{SCC} can then be removed.

The second idea of the PIE operation comes from the concept of the Π-edges described as follows.

Definition 7: Given a directed graph $G = (V, E)$ and an edge (u, v) in E, the edge (u, v) is a Π-edge if there also exists an edge (v, u) in E.

For illustration, in Fig. 3(a), all thin edges are the Π-edges. For the sake of the clarity, the set of the Π-edges in a graph is called PIE in the following discussions.

According to the definition of Π-edge, if an edge (u, v) is in the PIE, then there is a cycle $\langle u, v, u\rangle$ with length two. Please note that either u or v needs to be chosen to break this cycle, and the Π-edges, (u, v) and (v, u), must be removed after that. Therefore, the other cycles containing one of the Π-edges can be broken automatically. For example, the graph G in Fig. 3(a) is not a contractible graph and contains no acyclic edge. Because the thin edges in Fig. 3(a) are Π-edges, they must be removed to break the cycles formed by themselves. Consequently, the bold edges are acyclic edges in the remaining graph and they can be removed. After removing the bold edges, the remaining graph can be reduced as shown in Fig. 3(b). Therefore, the optimum solution $\{a, f\}$ can be found.

According to the concept of Π-edge, the acyclic edges in the graph G-PIE can be removed for the cycle breaking consideration of G. Theorem 2 and Corollary 2 formally state this observation.

Theorem 2: Given a directed graph $G = (V, E)$, the PIE of G and a cutset C of G, there is no edge $(u, v) \in$ PIE in the graph G–C.

Corollary 2: Given a graph $G = (V, E)$, the PIE of G and an edge (u, v) in the graph G-PIE, if there is no cycles in G-PIE containing (u, v), (u, v) can be removed for the cycle breaking consideration of G.

Given a graph $G = (V, E)$ and the PIE of G, the PIE operation deletes the acyclic edges of G-PIE by finding the component graph of G-PIE. Because the component graph can be found by the depth-first search, this operation runs on time complexity $O(|E| + |V|)$. The PIE operation defines a new class of graphs whose MFVS can be computed in polynomial time. We call this class *PIE-contractible*. The most important thing is that the class of PIE-contractible graphs can subsume the class of *scc-compressible graphs*, which was proposed in [7], while its

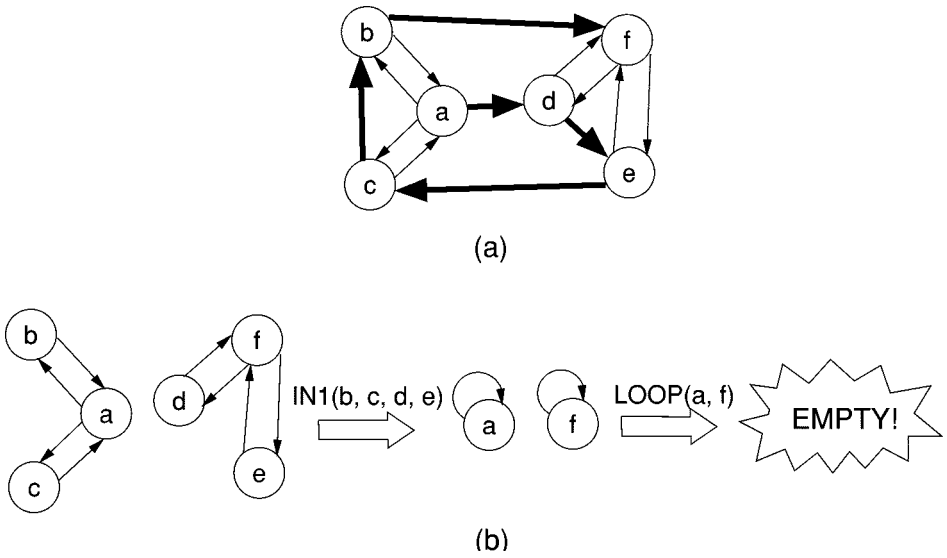

Fig. 3. (a) A noncontractible graph. (b) Graph reduction after PIE operation.

MFVS can be computed with the same time complexity as that of scc-compressible.

B. CORE Operation

The following definitions are necessary for further discussion.

Definition 8: Given a directed graph $G = (V, E)$ and the PIE of G, a vertex v is a *neighbor* of a vertex u if there is an edge $(u, v) \in$ PIE. Given a subgraph $G' = (V', E')$ of G where $V' \subseteq V$ and $E' \subseteq$ PIE $\subseteq E$, if each vertex in V' has no self-loop and it is the neighbor of the other vertices in V', G' is a *d-clique* of G.

Please note that if we remove all edges except the Π-edges and treat the pair of Π-edges (u, v) and (v, u) as an undirected edge (u, v), the d-clique can be viewed as the *clique* in an undirected graph.

Definition 9: Given a directed graph $G = (V, E)$, the PIE of G and a d-clique $G' = (V', E')$ of G, a vertex u is a CORE of G' if $u \in V'$ and all its incident edges are in E'. A vertex v is a Π-*vertex* if all its incident edges are in the PIE.

For illustration in Fig. 4, the vertex a and the vertex h are the neighbors of each other, but the vertex h is not a neighbor of the vertex c. The vertex set $\{a, b, h\}$ forms a d-clique and both the vertex a and the vertex b are the cores of the d-clique $\{a, b, h\}$. The vertices a, b, and d are the Π-vertices.

Given a d-clique $G' = (V', E')$ in a graph G, if a vertex v is a core of G', $CORE$ operation removes the vertex set $\{V' - v\}$ into the cutset C of G. The following discussion explains that the CORE operation does not impair the optimality of the cutset C.

Lemma 1: Given a d-clique $G' = (V', E')$ with n vertices, there are at least $n - 1$ vertices in any cutset of G'.

The graph G in Fig. 4(a) is not a PIE-contractible graph. According to Lemma 1, to break all cycles in the d-clique $\{a, b, h\}$, at least two vertices need removing and so do all edges incident to $\{a, b, h\}$. Please note that all incident edges of the core a are in the d-clique $\{a, b, h\}$. Removing all vertices in $\{a, b, h\}$ except the core a does not impair the optimality of final solution because there must be no cycle in the remaining graph containing the vertex a. After removing the vertices $\{b, h\}$ into the cutset, the remaining graph is contractible and the minimum cutset $\{b, h, d\}$ can be found as shown in Fig. 4(b). Theorem 3 states this observation.

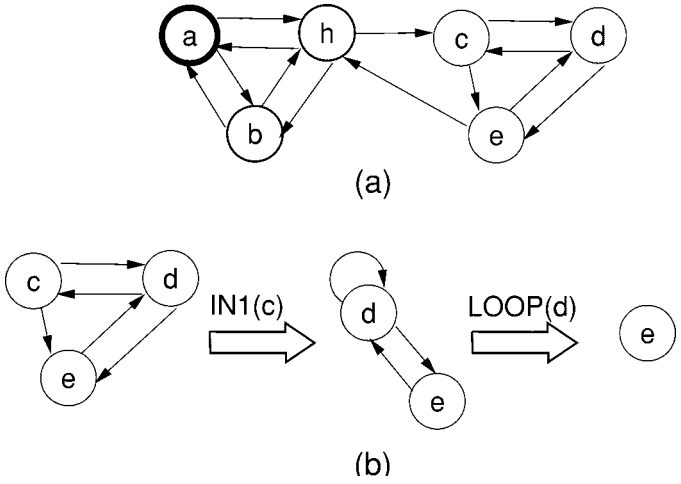

Fig. 4. (a) A graph is not the PIE-contractible. (b) Graph reduction after CORE operation.

Theorem 3: Given a directed graph $G = (V, E)$ and a core v of the d-clique $G' = (V', E')$ in G, breaking all cycles in G' by removing $\{V' - v\}$ into the cutset C of G does not impair the optimality of C.

Furthermore, if a core exists, it can be found by the CORE operation shown in Fig. 5 in time-complexity O($|E| + |V| \log |V|$) according to Theorems 4–6.

For the sake of the clarity, the set of the Π-vertices in a graph is called PIV in the following discussions.

Theorem 4: Given a directed graph $G = (V, E)$ and the PIV of G, a vertex v could be the core if it is in the PIV and has the minimum degree compared to all its neighbors.

Theorem 5: Given a directed graph $G = (V, E)$ and a Π-vertex v with its neighbors $N(v)$, the vertex v is a core of $G' = (\{N(v), v\}, E')$ if G' is a d-clique.

```
Cutset <- CORE(G, PIV, PIE){
1. sort(PIV);
   foreach v in PIV{
2.   if(v.valid == true){
       Is_d-clique = check_d-clique({v, N(v)});
3.     if(Is_d-clique)
         G = G - {v, N(v)}, return(N(v));
4.     else
         lable_invalid_core({v, N(v)});
}  } }
```

Fig. 5. The pseudocode of CORE operation.

Theorem 6: Given a directed graph $G = (V, E)$ and a Π-vertex u with the minimum degree n compared to its neighbors $N(u)$, if the vertex u is not a core, each vertex in $N(u)$ is not a core in G either.

The pseudocode of the CORE operation is shown in Fig. 5. According to Theorem 4, only the vertices in the PIV need to be checked. Therefore, given a graph $G = (V, E)$, the PIE and the PIV of G, the CORE operation first sorts the vertices in PIV in the ascending order according to the degree. Then, it chooses the first vertex v with "valid" mark in PIV and checks if $\{v, N(v)\}$ forms a d-clique by breadth-first search. If $\{v, N(v)\}$ forms a d-clique $G' = (V', E')$, v is a core of G' according to Theorem 5. Then, the vertices in V' and their incident edges are removed from G and the vertices $\{V' - v\}$ are placed into the cutset C. Otherwise, all vertices in $\{N(v), v\}$ are marked "invalid" according to Theorem 6 and it continuously checks the next vertex with "valid" mark in PIV. Because of the sorting and the breadth-first search procedures, the CORE operation runs on time-complexity $O(|E|+|V|\log|V|)$. The CORE operation also defines a new class of graphs whose MFVS can be computed in polynomial-time. We call this class *CORE-contractible*.

C. DOME Operation

The DOME (dominated edge) operation comes from the concept that only *minimal cycles* described in Definition 10 are necessary for cycle breaking consideration. Because no minimal cycle goes through the *dominated edges* described in Definition 12, the dominated edges can be removed.

Definition 10: A cycle $C_1 = \langle v_1, v_2, \ldots, v_n, v_1\rangle$ is said to be *covered* by a cycle $C_2 = \langle u_1, u_2, \ldots, u_k, u_1\rangle$ if each vertex v_i, $i = 1 \ldots n$, in C_1 is contained by C_2. A cycle is *minimal* if it does not cover another cycle.

For illustration in Fig. 6, the cycle $\langle a, c, a\rangle$ is a minimal cycle, but the one $\langle a, c, d, b, a\rangle$ is not.

Theorem 7: Given a directed graph G, only minimal cycles need to be broken while computing the MFVS of G.

Theorem 7 is trivial because breaking a minimal cycle C will automatically break all cycles covering C.

Definition 11: Given an edge (u, v), the vertex u is a *predecessor* of the vertex v and the vertex v is a *successor* of the vertex u. If (u, v) is a Π-edge, then the vertex u is called Π-*predecessor* of the vertex v and the vertex v is called Π-*successor* of the vertex u. Otherwise, the vertex u is called *non*-Π-*predecessor* of the vertex v and the vertex v is called *non*-Π-*successor* of the vertex u.

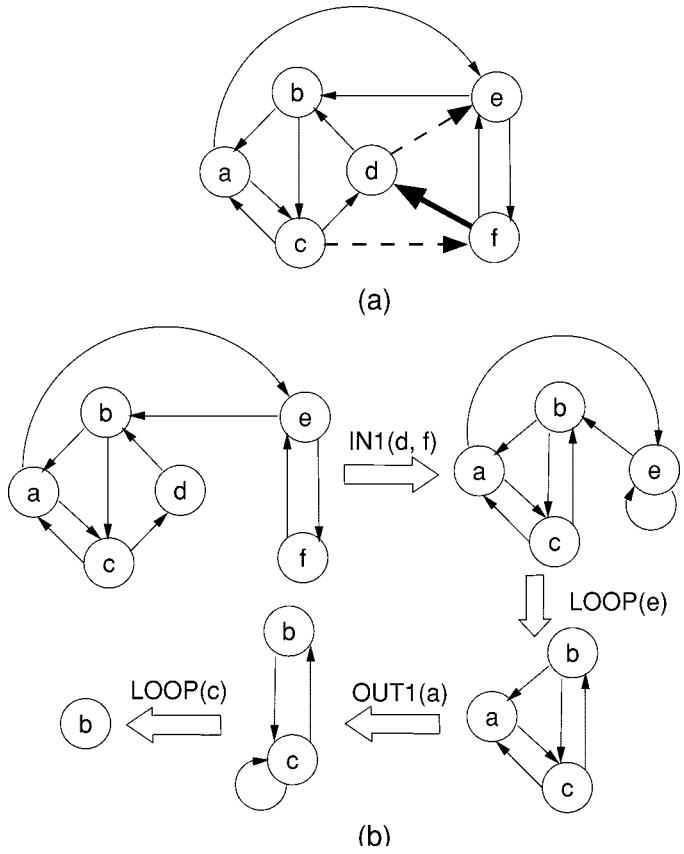

Fig. 6 (a) A graph is not CORE-contractible or PIE contractible and (b) graph reduction after DOME operation.

Definition 12: Given a graph G without self-loops, an edge (u, v) is *dominated* if one of the following conditions is satisfied. 1) $P_u \subseteq P_v$;P_u is the vertex set that consists of all *non*-Π-*predecessors* of the vertex u and P_v is the vertex set that consists of all *predecessors* of the vertex v. 2) $S_v \subseteq S_u$;$S_v$is the vertex set that consists of all *non*-Π-*successors* of the vertex v and S_u is the vertex set that consists of all *successors* of the vertex u. Otherwise, (u, v) is a *nondominated edge*.

In Fig. 6(a), the bold edges satisfy the condition 1 and the dashed edges satisfy the condition 2. Therefore, they are the *dominated edges*.

Theorem 8: Given a directed graph G without self-loops, only nondominated edges are necessary for computing the MFVS of G.

We use Fig. 7 to explain Theorem 8. In Fig. 7(a), P_u denotes the vertex set of the non-Π-predecessors of u, P_v denotes the vertex set of the predecessors of v and $P_u \subseteq P_v$. Therefore, (u, v) is a dominated edge. We claim that all cycles that contain the edge (u, v) are not the minimal ones. First, the cycles that contain the edge (u, v) must go through one of the predecessors of u. Without loss of generality, we assume that they are $C_1 = \langle x, u, v, w_1, w_2, \ldots, w_k, x\rangle$ and $C_2 = \langle P_{ui}, u, v, w_1, w_2, \ldots, w_k, P_{ui}\rangle$. The cycle C_1 goes through the Π-predecessor x of u and the cycle C_2 goes through the non-Π-predecessor of u. Because the vertex x is a Π-predecessor of u, the cycle C_1 must cover the cycle $\langle x, u, x\rangle$. Therefore, the cycle C_1 is not a minimal cycle. For the cycle C_2, because $P_u \subseteq P_v$, there

must be a cycle $C_3 = \langle P_{ui}, v, w_1, w_2, \ldots, w_k, P_{ui}\rangle$. Because $C_2 = \langle P_{ui}, u, v, w_1, w_2, \ldots, w_k, P_{ui}\rangle$ covers $C_3 = \langle P_{ui}, v, w_1, w_2, \ldots, w_k, P_{ui}\rangle$, it is not the minimal one. On the other hand, the condition 2 is symmetric to the condition 1 and its concept is shown in Fig. 7(b). Therefore, the dominated edges can be removed for cycle breaking consideration as stated in Theorem 8.

The graph in Fig. 6(a) is neither PIE-contractible nor CORE-contractible. However, both the bold and the dashed edges are the dominated ones and thus can be removed according to Theorem 8. After that, the remaining graph is the contractible graph and the minimum cutset $\{c, e\}$ can be found as shown in Fig. 6(b). For each edge, checking the satisfaction of the conditions 1 and 2 mentioned above can be conducted with time-complexity O($|V|$), so the DOME operation runs on O($|E||V|$) time-complexity. Therefore, it also defines a new class of graphs called *DOME-contractible* graphs whose MFVS can be found in polynomial-time.

IV. Our Exact Algorithm

Our exact algorithm based on the branch and bound strategy is shown in Fig. 8. This procedure is invoked at the top level with the following parameters: the graph $G = (V, E)$ to be solved, the current solution CC (which is the empty set initially), the best solution BC seen so far (which is V initially), a lower bound Min_P (which is zero initially), a tag L (which denotes the current recursive level) and a tag *reducible* (which denotes that if G could be further reduced and is one initially). It recursively reduces the size of G and returns the minimum cutset of G.

The eight contraction operations can be used to reduce the problem size without impairing the optimality. Another method to reduce the problem size is partitioning. The *strongly connected components* (SCC's) of a graph form disjoint subproblems for cycle breaking. In our algorithm, the graph reduction step is applied before partitioning. This is because the graph reduction step can remove redundant dependencies and vertices, and the remaining graph could be further partitioned into more SCC's.

At the graph reduction step, the eight contraction operations are applied to reducing the graph G iteratively until G can no longer be reduced. If G is empty after that, the minimum cutset of G is found. Otherwise, the number of SCC's in the reduced graph is calculated. If the number of SCC's is larger than 1, the cycle breaking procedure is applied to each SCC, respectively. Otherwise, both the *initial solution* and the *lower bound* of this reduced graph are calculated by the method described in Section IV-A and Section IV-B, respectively. The *lower bound* is the number of the vertices needed at least to break all cycles in the remaining graph. The search is bounded and terminated if the sum of the current solution and the lower bound is equal to or larger than the best solution seen so far. Otherwise, a vertex v with the maximum weight is selected as the branching vertex. The weight function will be described in Section IV-A. The branching is done by assuming that the vertex v is in the minimum cutset or not. If the vertex v is in the minimum cutset, the *Remove* operation appends it into the current solution CC and removes it and its incident edges from the graph. Otherwise, the *Merge* operation removes the vertex v from the graph and connects all predecessors of v to each successor of v. After branching, the current best solution is changed to the returned solution, if the returned solution is better.

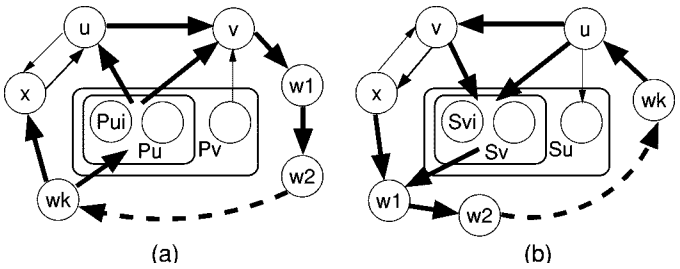

Fig. 7. The concept of the DOME operation.

A. Initial Solution Calculation

Our algorithm for calculating the initial solution is modified from the contraction algorithm mentioned in Section II-A. The modifications are listed as follows:

1) The operations, PIE, CORE, and DOME, are added into the graph reduction step.
2) The weight function for heuristic selection is modified as "$\alpha\times$(*number of* Π-*edges*)$+\beta\times$(in-degree plus out-degree)."
3) The heuristic is done by *merging* the vertex with the *minimum weight* rather than *removing* the vertex with the *maximum weight*. (Merging a vertex v means that remove the vertex v from the graph and connect all predecessors of v to each successor of v.)

The reason for the modification 2 is that if a vertex v has a Π-edge (u, v), there exists a cycle $\langle u, v, u\rangle$. Either u or v needs to be removed to break this cycle. It implies that v is highly likely to appear in the minimum cutset.

The reason for the modification 3 comes from the results of the following experiments. First, we define the following items.

Definition 13: A graph $G = (V, E)$ is complete if there exists the edges (v_i, v_j) in $G, \forall v_i, v_j \in V$ and $i \neq j$. The *density* of a graph $G = (V, E)$ is defined by "$|E|/|E_{\mathrm{comp}}|$;" $|E_{\mathrm{comp}}|$ is the number of the edges in the complete graph $G_{\mathrm{comp}} = (V, E_{\mathrm{comp}})$. A graph is said to be *nonreducible* if the eight contraction operations can no longer reduce it.

For different densities, we randomly produce ten *nonreducible* graphs with 100 vertices. Then, we apply the algorithms H8R (heuristic is based on removing the vertex with the maximum weight) and H8M (heuristic is based on merging the vertex with the minimum weight) to them, respectively. In Table I, the rows Heu_H8R and Heu_H8M denote the average times of the heuristic selection being conducted in solving a graph by the two algorithms, respectively, and the rows Sol_H8R and Sol_H8M denote the average size of the cutsets produced by the two algorithms, respectively.

We can find that algorithm H8R produces the cutsets whose sizes are close to the times of heuristic selection being conducted by it, but algorithm H8M does not, especially in the graphs with higher density. This is because a graph G with higher density is more likely a nonreducible graph. Removing

```
Cutset <- exact_cycle_break(G, CC, BC, Min_P, L, reducible){
  if( reducible ){
1./* graph reduction */
    CC = CC + graph_reduction(G);
2./* can graph be partitioned ? */
    if(numof(SCC(G)) > 1){
      foreach SCCi = (V', E') in G
        CC = CC + exact_cycle_break(SCCi, 0, V', 0, 0, 0);
      Return(CC);
  }}
3./* lower bound and initial solution calculation */
  lower_bound = bound_calculation(G);
  if(L == 0) {
    Min_P = lower_bound;
    BC = CC + initial_solution_calculation(G);
  }
4./* bounding the search ? */
  if( |CC| + lower_bound > |BC| ) return(BC);
  else if(G is empty) return(CC);
5./* branching by assuming v is in the minimum cutset or not */
  v = maximum_weight_vertex(G);
  Left = exact_cycle_break(Remove(G, v), CC+v, BC, Min_P, L+1, 1);
  if( |Left| < |BC| ) BC = Left;
  if( Min_P == |BC| ) return(BC); /* bounding the search */
  Right = exact_cycle_break(Merge(G, v), CC, BC, Min_P, L + 1, 1);
  if( |Right| < |BC| ) BC = right;
  return(BC);
}
```

Fig. 8. The pseudocode of our exact algorithm.

TABLE I
COMPARISON BETWEEN THE HEURISTICS H8R AND H8M

density	0.1	0.3	0.5	0.7	0.9
Heu_H8R	54.1	79.3	88.6	90.3	92.5
Sol_H8R	58.5	82.8	89.9	93.7	96.0
Heu_H8M	40.8	16	8.7	5.1	2.6
Sol_H8M	56.0	81.5	88.6	93.1	95.8

a vertex from G, the remaining graph is still most likely nonreducible because its density is still high. Therefore, the heuristic selection could be conducted again and again. However, merging a vertex in G could produce several self-loops and the remaining graph can be reduced without conducting heuristic selection. Therefore, the times of heuristic selection can be reduced significantly, especially in the graphs with higher density. The fewer times heuristic selections are conducted, the better solutions could be found. Because the nonreducible graph usually has a higher density, the algorithm H8M usually produces a better solution than the algorithm H8R.

B. Bounding Strategy

The lower bound calculation also uses the eight contraction operations. Similar to the contraction algorithm mentioned in Section II-A, it also contains the graph reduction and the heuristic selection. The heuristic is based on Lemma 1 mentioned in Section III-B, which we need at least $s-1$ vertices to break the cycles in a d-clique with size s, and we need at least one vertex to break a cycle. First, the graph reduction is applied iteratively until no more reduction is possible and the lower bound is updated according to the LOOP and the CORE operations. After that, if the graph is empty, the exact lower bound is found. Otherwise, the heuristic is applied as follows. 1) If a d-clique with the largest size s can be found, we remove it and its incident edges and then increase the lower bound by $s-1$. 2) If there is no d-clique with size being larger than one in the graph, we remove the cycle C with the minimum length and increase the lower bound by one. The graph reduction and the heuristic selection are iteratively applied until the graph is empty, and then the lower bound is returned. Because finding the largest d-clique in a directed graph is a NP-complete problem, which is equal to finding the largest clique in an undirected graph, the heuristic based on the concept of common neighbor is applied [22].

C. MCFVS Problem

Given a weighted directed graph $G = (V, E, W)$, where each vertex in V is associated with a cost w, find a cutset C such that the sum of the cost of the vertices in the cutset C is the minimum.

Our algorithm can be easily modified to cope with the MCFVS problem as follows. First, the sum of the costs of the vertices instead of the number of the vertices in the cutset C is used as the cost function. Second, the weight function of a vertex becomes ($\alpha\times$(number of Π-edges) $+\beta\times$(in-degree

+ out-degree))/(vertex cost). The lower bound estimation increases the lower bound by the sum of the costs of the vertices in the removed d-clique except the one with the maximum cost or by the minimum cost of the vertices in the removed cycle. Last, the contraction operations IN1, OUT1, and CORE are modified as follows. The cost of the unique predecessor v' of a vertex v must be smaller than that of the vertex v for the IN1 operation. The cost of the unique successor v' of a vertex v must be smaller than that of the vertex v for the OUT1 operation. The core of a d-clique must have the maximum cost in this d-clique for the CORE operation.

V. Experimental Results

To demonstrate the efficiency of our algorithm, we have implemented our algorithm and applied it to solving the partial scan problem in ISCAS'89 benchmarks. In our experiments, the self-loops in the initial S-graph are ignored first, since these loops do not cause problems during test generation [8]. The following experiments are run at SUN-UltraII workstation with 256M RAM.

In Table II, the algorithm H5R is the previous contraction algorithm proposed in [10] in which the "in-degree + out-degree" is used as the weight function. The algorithm H5WR modifies the weight function of H5R to "$\alpha\times$(number of Π-edges) $+\beta\times$(in-degree + out-degree)" where $\alpha = 1$ and $\beta = 1$. The sizes of the cutsets produced by the two algorithms are shown in columns "H5R" and "H5WR," respectively. The sizes of exact cutsets are shown in column "*exact*." In all circuits in Table II, the algorithm H5WR does not produce worse results than the algorithm H5R. Both the algorithms H5R and H5WR produce the exact cutset for the circuits without being highlighted except s1423 and s15850. Furthermore, the algorithm H5WR produces better results for the circuits highlighted, especially for s38584. The circuits in ISCAS'89 benchmarks not being listed in Table II can be fully reduced by the algorithm H5R, so we do not need to consider them in our experiments.

TABLE II
Weight Function Comparison

	graph size	H5R	H5WR	exact
s13207	669	59	59	59
s1423	74	22	22	21
s1488	6	5	5	5
s1494	6	5	5	5
s15850	597	91	91	88
s344	15	5	5	5
s349	15	5	5	5
s382	21	9	9	9
‡ **s38417**	1636	**390**	**388**	**374**
‡ **s38584**	1452	**315**	**301**	**292**
s386	6	5	5	5
s400	21	9	9	9
s444	21	9	9	9
s510	6	5	5	5
s5378	164	30	30	30
s820	5	4	4	4
s832	5	4	4	4
‡ **s9234**	228	**54**	**53**	**53**
s953	6	5	5	5

The experimental results in Table III show that our new contraction operations are useful for the practical CAD application. In Table III, the algorithm H8WR adds three new contraction operations (PIE, CORE, and DOME) into the algorithm H5WR. The algorithm "H5WR + SCC" adds the SCC (strongly connected component) operation proposed in [2] and [7] into the algorithm H5WR. The sizes of the graphs that cannot be further reduced without conducting the heuristic selection, the times of heuristic selection conducted before the graph is fully reduced, the sizes of the cutsets produced and the CPU time consumed by these two algorithms are shown in columns "r_size," "heu_sel," "sol," and "$time$," respectively. The zeros in columns "r_size" and "heu_sel" mean that the corresponding S-graphs can be fully reduced, and the optimal cutsets can be found after applying the graph reduction. If only the SCC operation and five contraction operations (IN0, OUT0, IN1, OUT1, and LOOP) are applied, all circuits in Table III cannot be fully reduced. However, after adding our operations (PIE, CORE and DOME), the S-graphs can be fully reduced in 15 out of 19 circuits. In those 4 circuits (s1423, s15850, s38417, and s38584) cannot be fully reduced, the graph size can also be further reduced significantly. Therefore, our new contraction operations can improve the speed of the exact algorithms significantly. Also, the sizes of the cutsets produced by the algorithm H8WR are not larger than those produced by the algorithm H5WR + SCC. In particular, the algorithm H8WR outperforms in the circuits s1423, s15850, s38417, and s38584. The CPU time consumed by the algorithm H8WR only increases slightly. It is surprising that the algorithm H8WR runs faster than the algorithm H5WR + SCC in the circuits s15850, s38417, and s38584. The reasons could be that before a heuristic is conducted, the contraction operations must be applied with no gain returned, and the number of times of the heuristics being conducted by the algorithm H5WR + SCC is much larger than that of the algorithm H8WR.

In Table IV, the algorithm H8WM conducts the heuristics by merging the vertex with the minimum weight instead of the one adopted by the algorithm H8WR that removing the vertex with the maximum weight. Only nonreducible circuits need to be considered in this comparison. Though in the circuit s15850, the algorithm H8WM produces a solution larger than the algorithm H8WR by 1. However, in the circuit s38417, the algorithm H8WM produces a solution smaller than the algorithm H8WR by 8. The most important thing is that the algorithm H8WM makes no more heuristics than the algorithm H8WR for all circuits. Therefore, conducting merging instead of removing heuristic should produce better cutsets in most cases.

To further demonstrate the efficiency of the algorithm H8WM, we compare the results produced by the algorithm H8WM with the results produced by the exact algorithm for the graphs generated randomly. In Table V, column one and row one show the sizes and the densities of input graphs, respectively. The rows "H8WM" and "*Exact*" show the average sizes of the feedback vertex sets (FVS) produced by the algorithm H8WM and the exact algorithm. The rows "*approx. f*" show the average approximation factors of the algorithm H8WM at different graphs. The approximation factor is the ratio of the

TABLE III
CONTRACTION OPERATIONS COMPARSION

	H5WR+SCC				H8WR			
	r_size	heu_sel	sol	time(s)	r_size	heu_sel	sol	time(s)
s13207	47	26	59	0.24	**0**	**0**	59	0.24
s1423	53	14	**22**	0.04	26	7	**21**	0.04
s1488	6	4	5	0.01	**0**	**0**	5	0.01
s1494	6	4	5	0.01	**0**	**0**	5	0.01
s15850	176	57	**91**	**0.63**	28	9	**88**	**0.56**
s344	3	1	5	0.01	**0**	**0**	5	0.01
s349	3	1	5	0.01	**0**	**0**	5	0.01
s382	12	6	9	0.01	**0**	**0**	9	0.01
s38417	465	323	**386**	**3.27**	120	64	**382**	**2.04**
s38584	796	133	**301**	**2.08**	16	1	**292**	**1.18**
s386	6	4	5	0.01	**0**	**0**	5	0.01
s400	12	6	9	0.01	**0**	**0**	9	0.02
s444	12	6	9	0.01	**0**	**0**	9	0.01
s510	6	4	5	0.01	**0**	**0**	5	0.01
s5378	8	1	30	0.04	**0**	**0**	30	0.05
s820	5	3	4	0.01	**0**	**0**	4	0.01
s832	5	3	4	0.01	**0**	**0**	4	0.01
s9234	94	34	53	0.08	**0**	**0**	53	0.08
s953	6	4	5	0.01	**0**	**0**	5	0.01

TABLE IV
HEURISTIC COMPARISON AT ISCAS'89 CIRCUITS

	H8WR			H8WM		
	sol	heu_sel	time(s)	sol	heu_sel	time(s)
s1423	21	**7**	0.03	21	**2**	0.04
s15850	**88**	**9**	0.56	**89**	**4**	0.58
s38417	**382**	**64**	2.04	**374**	**8**	1.92
s38584	292	**1**	1.18	292	**1**	1.37

size of the FVS produced by the algorithm H8WM to the size of the FVS produced by the exact algorithm. In Table V, we can see that the results of the algorithm H8WM are very close to the ones of the exact algorithm and all the approximation factors are smaller than 1.1. Please note that the approximation factors are smaller in the graphs with very low density and the graphs with higher density. The reasons are the following: in the graphs with lower density, the contraction operations can largely reduce the sizes of graphs and thus result in better solutions. In the graphs with higher density, conducting the heuristic by merging a vertex results in fewer times of heuristic selections as mentioned in Section IV-A. The fewer times the heuristic selections are conducted, the better solutions could be found.

In the branch and bound algorithms, two factors can improve the speed. The one is a good initial solution and the other is a tight lower bound. The algorithm H8WM produces a good initial solution for our branch and bound algorithm. To demonstrate the effectiveness of our bounding strategy, we compare the algorithm of our lower bound calculation mentioned in Section IV-B with the one suggested in [23] and the one proposed in [7].

In [23], the *maximum independent set* of *independency graph* is suggested as a bounding strategy to solve the *column-covering* problem (see [23] for details). In MFVS problem, the maximum independent set can be used to compute the lower bound as shown below. First, all minimal cycles are enumerated. Each minimal cycle i is associated with a vertex i in the independency graph, which is an undirected graph. The edge (i, j) exists in the independency graph if all vertices in the cycle i are different from the vertices in the cycle j. Then, a lower bound for breaking all cycles in a graph can be computed by finding the maximum clique size in the independency graph. Because the maximum clique problem is a NP-complete problem, the good heuristic proposed in [22] is used in our experiments.

The authors of [7] use the *integer linear programming* (ILP) to model the MFVS problem. A lower bound can be calculated by relaxing their ILP formulation to *linear programming* (LP) formulation. To enhance the lower bound of this relaxation, they suggest adding the constraints caused by the cycles with length 2 (see [7] for details).

We conduct the experiments as follows. First, the graph reduction is applied to the S-graph of each circuit. After the graph reduction, we calculate the lower bounds for the remaining graphs, which are nonreducible. In Table VI, the columns two and three show the graph size before reduction and after reduction, respectively. The column "exact" shows the exact lower bounds. The lower bounds produced by the maximum independent set, the ILP relaxation and our approach are shown in columns "MIS," "ILP," and "Ours," respectively. The last column is the CPU time consumed by our approach (including the CPU time consumed by the graph reduction). We can find that for the circuit s38417, both approaches "MIS" and "ILP" produce a lower bound less than the exact one by 28. However, our approach finds the exact lower bound for all circuits in Table VI in very short CPU time. Because we perform the "ILP" approach by running the commercial tool LINGO on 586

TABLE V
RESULT COMPARISON BETWEEN THE H8WM ALGORITHM AND THE EXACT ALGORITHM AT THE GRAPHS PRODUCED RANDOMLY

size	density	0.0125	0.025	0.05	0.1	0.3	0.5	0.7	0.9
50	H8WM	0.5	4.5	11	19.2	34.7	40.5	44.1	46.2
	Exact	0.5	4.5	10.8	17.9	33.5	39.8	43.3	46
	approx. f	**1**	**1**	**1.02**	**1.08**	**1.04**	**1.02**	**1.02**	**1.01**
60	H8WM	0.5	7.6	15.4	24.5	44	50.2	53.4	56.2
	Exact	0.5	7.5	14.9	23.1	42.6	49.3	53	56
	approx. f	**1**	**1.02**	**1.04**	**1.06**	**1.04**	**1.02**	**1.01**	**1.01**
70	H8WM	1.3	11.1	21.2	30	52.8	59.7	63.6	66.2
	Exact	1.3	11	19.8	28	51.3	59	62.9	65.7
	approx. f	**1**	**1.01**	**1.07**	**1.08**	**1.03**	**1.02**	**1.02**	**1.02**
80	H8WM	1	13.4	26.5	38.9	62.2	69.5	73.4	76
	Exact	1	13	24.7	36.8	60.2	68.2	72.6	75.8
	approx. f	**1**	**1.03**	**1.08**	**1.06**	**1.04**	**1.02**	**1.02**	**1.01**
90	H8WM	1.4	18.7	33.6	46.6	71.6	79.9	83.5	86.1
	Exact	1.4	17.9	31	44.1	70	78	82.2	85.6
	approx. f	**1**	**1.05**	**1.09**	**1.06**	**1.03**	**1.03**	**1.02**	**1.01**

PC and enumerating all minimal cycles consumes much CPU time, the CPU time consumed by the approaches "MIS" and "ILP" are not listed here. Furthermore, we randomly produce ten nonreducible graphs with 50 vertices for the different density listed in Table VII and apply the three bounding strategies to them. For these graphs, our approach also produces better lower bounds than the other two approaches. An interesting phenomenon is that when the graph density is high, it seems that the approaches "MIS" and "ILP" could not produce a lower bound higher than 25 for these graphs. However, our approach still produces good lower bounds for them. Therefore, our bounding strategy can prune the solution space efficiently and enhances the speed of our exact algorithm.

Finally, we summarize the CPU time consumed and the results produced by our exact algorithm in Table VIII. It is surprising that for all circuits in ISCAS'89 benchmarks our exact algorithm can find out the exact cutsets in less than 3 s (CPU time).

TABLE VI
LOWER BOUND COMPARISON AT ISCAS'89 CIRCUITS

	size	r_size	exact	MIS	ILP	Ours	time(s)
s1423	74	26	11	9	10.5	11	0.04
s15850	597	28	13	13	13	13	0.59
s38417	1636	120	**88**	**60**	**60**	**88**	1.90
s38584	1452	16	5	4	4.5	5	1.18

TABLE VII
LOWER BOUND COMPARISON AT THE GRAPHS PRODUCED RANDOMLY

density	0.1	0.3	0.5	0.7	0.9
MIS	13.8	22.3	24.2	24.8	24.9
ILP	10.23	24.85	25	25	25
Ours	15	26.4	32.7	38.2	43.7
exact	17.9	33.5	39.8	43.3	46.0

VI. CONCLUSION

The contributions of this paper are summarized as follows. 1) Three new and powerful contraction operations that can further reduce a graph are developed for computing the MFVS of a graph. These contraction operations define a new class of graphs called reducible graphs whose MFVS can be found in polynomial-time complexity. Furthermore, the contraction operations can be incorporated with the existent exact algorithms and the approximation algorithms to improve the speed of the exact algorithms and to reduce the value of approximation factors of the approximation algorithms. 2) An algorithm based on branch and bound strategy is developed for solving the MFVS problem exactly. This algorithm employs a heuristic that produces a good initial solution and a good bounding strategy to prune the solution space. We have implemented our algorithms and applied them to solving the partial scan problem in ISCAS'89 benchmarks. In these experiments, our new contraction operations can significantly enhance not only the quality of the FVS produced by our heuristic algorithms but also the speed of our exact algorithm. The experimental results further show that our exact algorithm can find out the exact cutsets for all ISCAS'89 benchmarks in less than 3 s (CPU time) on SUN-UltraII workstation. Therefore, the new contraction operations and our algorithms are demonstrated to be very effective in the partial scan application.

APPENDIX

Corollary 1: Given a graph $G = (V, E)$ and its corresponding component graph $G^{\mathrm{SCC}} = (V^{\mathrm{SCC}}, E^{\mathrm{SCC}})$, the edges in E associated with an edge in E^{SCC} do not belong to any cycle in G and can be eliminated from G for the cycle breaking consideration.

Proof: Assume that an edge (u, v) in E is associated with an edge in E^{SCC} and the edge (u, v) also belongs to a cycle in G. Because (u, v) is associated with an edge in E^{SCC}, the vertices u and v must be in the different component vertices of G^{SCC}. However, the edge (u, v) belongs to a cycle in G, so u and v are reachable from each other. They must be in the same strongly connected component. It is contradictive. Therefore, no cycle in

TABLE VIII
THE RESULTS PRODUCED BY OUR EXACT ALGORITHM

circuit	sol.	time(s)	circuit	sol.	time(s)	circuit	sol.	time(s)	circuit	sol.	time(s)
s1196	0	**0.01**	s27	1	**0.01**	s386	5	**0.01**	s641	7	**0.01**
s1238	0	**0.01**	s298	1	**0.01**	s400	9	**0.01**	s713	7	**0.01**
s13207	59	**0.24**	s344	5	**0.01**	s420	0	**0.01**	s820	4	**0.01**
s1423	21	**0.06**	s349	5	**0.01**	s444	9	**0.01**	s832	4	**0.01**
s1488	5	**0.01**	s35932	306	**1.30**	s510	5	**0.01**	s838	0	**0.01**
s1494	5	**0.01**	s382	9	**0.01**	s526	3	**0.01**	s9234	53	**0.07**
s15850	88	**0.64**	s38417	374	**2.03**	s526n	3	**0.01**	s953	5	**0.01**
s208	0	**0.01**	s38584	292	**1.18**	s5378	30	**0.03**			

G contains the edge (u, v) and the edge (u, v) can be eliminated for the cycle breaking consideration.

Theorem 2: Given a directed graph $G = (V, E)$, the PIE of G and a cutset C of G, there is no edge $(u, v) \in$ PIE in the graph G–C.

Proof: Assume that an edge (u, v) in the graph G–C is in the PIE. According to the definition of Π-edge, there exists a cycle $\langle u, v, u\rangle$ in the graph G–C. Therefore, the graph G–C is cyclic. It is contradictive.

Corollary 2: Given a graph $G = (V, E)$, the PIE of G and an edge (u, v) in the graph *G-PIE*, if there is no cycles in *G-PIE* containing (u, v), (u, v) can be removed for the cycle breaking consideration of G.

Proof: Assume that there is no cycle in the graph *G-PIE* containing the edge (u, v). If some cycles in G contain (u, v), those cycles must contain some edges in the PIE. According to Theorem 2, those cycles must be broken after the edges in the PIE are removed. Therefore, those cycles do not need to be considered for cycle breaking and the edge (u, v) can be removed.

Lemma 1: Given a d-clique $G' = (V', E')$ with n vertices, there are at least $n - 1$ vertices in any cutset of G'.

Proof: Assume that C is a cutset of G' and the number of the vertices in C is less than $n - 1$. Therefore, the graph G'–C contains two vertices, u and v, at least. By the definition of d-cliques, there exists a Π-edge (u, v) in G'–C. Therefore, there is a cycle in the graph G'–C. It is contradictive.

Lemma 2: Given a d-clique $G' = (V', E')$ with n vertices, randomly choosing $n-1$ vertices from V' forms a cutset of G'.

Proof: Because G' is a d-clique, each vertex in V' has no self-loop. A graph with a single vertex without self-loops is acyclic.

Theorem 3: Given a directed graph $G = (V, E)$ and a core v of the d-clique $G' = (V', E')$ in G, breaking all cycles in G' by removing $\{V'$–$v\}$ into the cutset C of G does not impair the optimality of C.

Proof: According to Lemma 1 and Lemma 2, to break all cycles in G', only one vertex can be reserved and all edges in G' must be removed. Please note that all incident edges of the core v are in G'. This means that after removing $\{V'$–$v\}$, there is no cycle containing v in the remaining graph. Therefore, breaking all cycles in G' by removing $\{V'$–$v\}$ into C does not impair the optimality of the final solution.

Lemma 3: Given a directed graph $G = (V, E)$ and a d-clique $G' = (V', E')$ of G, if there exists a core v of G' the core v has the minimum degree in V'.

Proof: Given a vertices u with the minimum degree n in V' and assume that the core v has degree $m, m > n$. Because v is a core of G', all its incident edges are associated with its neighbors. Because both v and u are in the d-clique G', the neighbors of v are also the neighbors of u. Therefore, the degree of u is not smaller than the degree of v, i.e., $n \geq m$. It is contradictive.

Lemma 4: If a vertex u is a core of a d-cliques $G' = (V', E')$ in G u is a Π-vertex.

Proof: According to Definition 8 and 9, all edges in E' belong to PIE and the edges of a core must belong to E'. Therefore, the vertex u must be a Π-vertex.

Theorem 4: Given a directed graph $G = (V, E)$ and the PIV of G, a vertex v could be the core if it is in the PIV and has the minimum degree compared to all its neighbors.

Proof: It is obvious from Lemma 3 and Lemma 4.

Theorem 5: Given a directed graph $G = (V, E)$ and a Π-vertex v with its neighbors $N(v)$, the vertex v is a core of $G' = (\{N(v), v\}, E')$ if G' is a d-clique.

Proof: The $\{N(v), v\}$ forms a d-clique and all incident edges of v are associated with its neighbors. Therefore, the vertex v is a core of the d-clique $G' = (\{N(v), v\}, E')$.

Theorem 6: Given a directed graph $G = (V, E)$ and a Π-vertex u with the minimum degree n compared to its neighbors $N(u)$, if the vertex u is not a core, each vertex in $N(u)$ is not a core in G either.

Proof: Assume that the vertex u is not a core and a core v is the neighbor of u. According to Theorem 4 and Theorem 5, the core v must also have a degree n and the $\{N(v), v\}$ forms a d-clique G'. Because the vertex u is a neighbor of v, the vertex u is also in this d-clique G'. Furthermore, because u has a degree n, all incident edges of u must also be in G'. Therefore, the vertex u is also a core according to the definition of core. It is contradictive.

Theorem 7: Given a directed graph G, only minimal cycles need to be broken while computing the MFVS of G.

Proof: For each cycle C which is not minimal, there exists a minimal cycle C_m covered by the cycle C according to Definition 10. Because breaking C_m automatically breaks the cycle C only minimal cycles need to be considered for computing the MFVS of a directed graph.

Lemma 5: Given a directed graph $G = (V, E)$, an edge $(u, v) \in E$ and a vertex x that is a Π-predecessor of the vertex u, the cycle C containing the path $\langle x, u, v\rangle$ is not minimal.

Proof: Because x is a Π-predecessor of the vertex u, there must exist a cycle $\langle x, u, x\rangle$ with length two covered by C.

Lemma 6: Given a directed graph $G = (V, E)$, an edge $(u, v) \in E$ and a vertex x that is a Π-successor of the vertex v, the cycle C containing the path $\langle u, v, x\rangle$ is not minimal.

Proof: Lemma 6 is symmetric to Lemma 5.

Lemma 7: Given a directed graph $G = (V, E)$ without self-loops, an edge $(u, v) \in E$, the sets P_u and P_v that consist of the predecessors of the vertices u and v, respectively, if $P_u \subseteq P_v$, then each cycle contains the edge (u, v) is not minimal.

Proof: Without loss of generality, we assume that the vertex $P_{ui} \in P_u$, and a cycle C_1 contains the edge (u, v). Because the cycle C_1 contains (u, v), it must go through one of the predecessors of u. Without loss of generality, we assume the predecessor is the vertex P_{ui} and $C_1 = \langle P_{ui}, u, v, w_1, w_2, \ldots, w_k, P_{ui}\rangle$; $(w_{i-1}, w_i) \in E$ and $i = 2, 3, \ldots, k$. If $P_u \subseteq P_v$, there must exist an edge (P_{ui}, v). Therefore, there also exist a cycle $C_2 = \langle P_{ui}, v, w_1, w_2, \ldots, w_k, P_{ui}\rangle$; $(w_{i-1}, w_i) \in E$ and $i = 2, 3, \ldots, k$. Because the cycle C_1 covers the cycle C_2, it is not minimal.

Lemma 8: Given a directed graph $G = (V, E)$ without self-loop, an edge $(u, v) \in E$, the sets S_v and S_u that consist of the successors of the vertices v and u, respectively, if $S_v \subseteq S_u$, then each cycle contains the edge (u, v) is not minimal.

Proof: Lemma 8 is symmetric to Lemma 7.

Theorem 8: Given a directed graph G without self-loops, only nondominated edges are necessary for computing the MFVS of G.

Proof: It is obvious from Theorem 7 and Lemmas 5–8.

REFERENCES

[1] H. Levy and D. W. Low, "A contraction algorithm for finding small cycle cutsets," *J. Algorithm*, vol. 9, pp. 470–493, 1988.

[2] G. W. Smith and R. B. Walford, "The identification of a minimal feedback vertex set of a directed graph," *IEEE Trans. Circuits Syst.*, vol. CAS-22, pp. 9–14, Jan. 1975.

[3] A. Xie and P. A. Beerel, "Accelerating Markovian analysis of asynchronous systems using string-based state compression," in *Proc. 4th Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, 1998, pp. 247–260.

[4] E. Trischler, "Incomplete scan design with an automatic test generation methodology," in *Proc. Int. Test Conf.*, Nov. 1980, pp. 153–162.

[5] J. Y. Jou and M. C. Nien, "Power driven partial scan," in *Proc. Int. Conf. Computer Design*, 1997, pp. 642–647.

[6] D. Kagaris and S. Tragoudas, "Retiming-based partial scan," *IEEE Trans. Comput.*, vol. 45, pp. 74–87, Jan. 1996.

[7] S. T. Chakradhar, A. Balakrishnan, and V. D. Agrawal, "An exact algorithm for selecting partial scan flip-flops," in *Proc. 31st Design Automation Conf.*, 1994, pp. 81–86.

[8] K. T. Cheng and V. D. Agrawal, "A partial scan method for sequential circuits with feedback," *IEEE Trans. Computer*, vol. 39, pp. 544–548, Apr. 1990.

[9] R. Ashar and S. Malik, "Implicit computation of minimum-cost feedback-vertex sets for partitial scan and other applications," in *Proc. 31st Design Automation Conf.*, 1994, pp. 77–80.

[10] D. H. Lee and S. M. Reddy, "On determining scan flip-flops in partial scan designs," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1990, pp. 322–325.

[11] J. Y. Jou and K. T. Cheng, "Timing-driven partial scan," *IEEE Design & Test of Computers*, pp. 52–59, Winter 1995.

[12] M. R. Garey and D. S. Johnson, *Computer and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.

[13] M. Yannakakis, "Node-deletion problem on bipartite graphs," *SIAM J. Comput.*, vol. 10, pp. 310–327, 1981.

[14] T. Leighton and S. Rao, "Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms,", unpublished.

[15] G. Even, J. Naor, B. Schieber, and M. Sudan, "Approximating minimum feedback sets and multicuts in directed graphs," *Algorithmica*, vol. 20, no. 2, pp. 151–174, 1998.

[16] R. Bar-Yehuda, D. Geiger, J. Naor , and R. M. Roth, "Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and bayesian inference," *SIAM J. Computing*, vol. 27, no. 4, pp. 942–959, 1998.

[17] C. Wang, E. L. Lloyd , and M. L. Soffa, "Feedback vertex sets and cyclically reducible graphs," *J. ACM*, vol. 32, no. 2, pp. 296–313, 1985.

[18] C. L. Lu and C. Y. Tang, "A linear-time algorithm for the weighted feedback vertex problem on interval graphs," *Inform. Processing Lett. 61*, pp. 107–111, 1997.

[19] Y. D Liang, "On the feedback vertex set problem in permutation graphs," *Inform. Processing Lett. 52*, pp. 123–129, 1994.

[20] K. T. Cheng and V. D. Agrawal, "Designing circuits with partial scan," *IEEE Design Test Comput.*, vol. 5, pp. 8–15, Apr. 1988.

[21] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. New York: McGraw Hill, 1990.

[22] C. J. Tseng and D. P. Siewiorek, "Automated synthesis of data paths in digital systems," *IEEE Trans. Computer-Aided Design*, vol. CAD-5, pp. 379–395, July 1986.

[23] R. L. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 727–750, Sept. 1987.

Hen-Ming Lin was born on June 30, 1972 in Taiwan, ROC. He received the B.S. degree in electronics engineering from the National Chiao Tung University, Hsinchu, Taiwan, ROC, in 1995. He is currently working toward the Ph.D. degree in the Department of Electronics Engineering at the same university.

His research interests include hardware description language (HDL) synthesis, graph theory, embedded software compilation, and high-level/logic synthesis.

Jing-Yang Jou received the B.S. degree in electrical engineering from National Taiwan University, Taiwan, R.O.C., and the M.S. and Ph.D. degrees in computer science from University of Illinois at Urbana-Champaign, in 1979, 1983, and 1985, respectively.

He is a Professor of Electronics Engineering at National Chiao Tung University, Taiwan, R.O.C. Before joining Chiao Tung University, he was with AT&T Bell Laboratories, Murray Hill, NJ. His research interests include behavioral and logic synthesis, VLSI designs and CAD for low power, design verification, and hardware/software codesign.

Dr. Jou has been elected to Tau Beta Pi. He was the recipient of the Distinguished Paper Award of the IEEE International Conference on Computer-Aided Design in 1990. He served as the Technical Program Chair of the Asia-Pacific Conference on Hardware Description Languages (APCHDL'97) and has published more than 80 journal and conference papers.