NORTH-HOLLAND

An Enhanced Zero-One Optimal Path Set Selection Method

Chyan-Goei Chung and Jen-Gaw Lee

Department of Computer Science and Information Engineering, National Chiao Tung University, Hsin-Chu, Taiwan, Republic of China

One issue in structural program testing is how to select a mimimal set of test paths to meet certain test requirements. This is referred to as the optimal path set selection problem. The previously proposed zero-one optimal path set selection method is a generalized method that can be applied to most coverage criteria. However, the major drawback of this method is that for a large program the computation may take ten or more hours because the computation is exponentially proportional to the number of candidate paths and proportional to the number of components to be covered. To alleviate the drawback, this paper enhances the method by (1) defining five reduction rules, and (2) reusing previously selected path sets to reduce both the number of candidate paths and the number of components to be covered. Since both the number of candidate paths and the number of components to be covered are reduced, the computation time can be greatly reduced. In addition, an efficient approach to handling infeasible paths is proposed. An evaluation of the enhanced zero-one method, the original zero-one method, and a greedy method is executed and the result is presented. © 1997 Elsevier Science Inc.

1. INTRODUCTION

Structural testing is a well-developed technique applied in program validation. In structural testing, program structure is mapped to a directed graph in which a node represents a code segment while a branch directs the transfer of control flow. Without loss of generality, it is assumed a sole source node and a sole terminal node both exist in the digraph. A path, starting from the source node and ending with the terminal node, is a sequence of nodes each connected by branches. Using network methodologies (Beizer, 1984) (McCabe, 1976), the complete path set (i.e., the set containing all paths) can then be constructed. Due to conditional branches and loops, even a small program may have many paths. Thus, all paths testing may be impractical. Instead of all paths testing, a subset of the complete path set, called a path set, is selected to satisfy the required coverage criterion. Assuming that the cost of testing each path is the same, the path set satisfying the required coverage criterion with minimal number of paths is desired. Finding a path set satisfying a required coverage criterion with minimal number of paths is referred to as *optimal path set selection problem.*

It is desired that the minimizing test path set does not impair the coverage criterion's ability to detect faults. In fact, in a recent empirical study (Wong, et al., 1995), the effect of reducing the size of a test path set on fault detection, while holding coverage constant, was analyzed with regard to the all-uses coverage criterion. For the program analyzed, the minimizing test path set produced very little or no reduction at all in fault detection effectiveness.

Many approaches (Krause et al., 1973) (Miller et al., 1974) (Ntafos and Hakimi, 1979) (Prather and Myers, 1987) (Wang et al., 1989) (Hsu and Chung, 1992) to finding a path set satisfying a required coverage criterion have been proposed. The zero-one optimal path set selection method (Wang et al., 1989) and the minimum flow method (Ntafos and Hakimi, 1979) are the only two that guarantee the obtained path set is optimal because they are developed on rigid mathematical grounds instead of heuristic rules. Comparing these two methods, the zero-one optimal path set selection method is more powerful than the minimum flow method because it

Address correspondence to Chyan-Goei Chung, Dept. of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan, Republic of China. e_mail: {cgchung, peterpan}@simm3.csie.nctu.edu.tw

J. SYSTEMS SOFTWARE 1997; 39:145–164

655 Avenue of the Americas, New York, NY 10010

0164-1212/97/$17.00
PII S0164-1212(96)00169-0

can be applied to a large variety of constraints, cost functions, and coverage requirements while the minimum flow method can be applied to all-nodes and all-branches coverage criteria only (Lin and Chung, 1989).

The major drawback of the zero-one integer programming method is that the required computation time is too long. The computation time of the zero-one optimal path set selection method is proportional to $(2^{|Path|} \times |Component|)$ where $|Path|$ represents the number of candidate paths while $|Component|$ represents the number of components to be covered. Obviously, the computation time can be reduced if the number of candidate paths and the number of components to be covered are reduced. Reducing the number of components is equal to decreasing the number of constraints while reducing the number of paths is equal to decreasing the number of decision variables. A component can be removed if and only if the corresponding constraint is redundant. Similarly, a candidate path can be removed if and only if whether the candidate path should be selected or not can be predetermined. Based on the concept, this paper proposes: (1) five reduction rules, and (2) reusing previously selected path sets to reduce both the number of candidate paths and the number of components to be covered.

For the infeasible path problem, (Lin and Chung, 1989) simply delete the infeasible paths from the complete path set, regenerate the coverage frequency matrix, reformulate the corresponding zero-one integer programming model, and then recompute to obtain another new optimal path set. The drawback is that it takes too much computation time. This paper proposes another approach which reuses previously selected feasible paths to reduce the long computation.

Minimization is only worthwhile if the reduction in cost derived from reducing the path set compensates for the cost of doing the minimization. Otherwise, the greedy method (Hsu and Chung, 1992) which obtains near optimal path set with less computation time, is more practical. To evaluate the enhanced zero-one method, the original zero-one method, and a greedy method, four experiments have been executed and the result is presented in this paper.

This paper is organized as follows. Section 2 introduces the zero-one optimal path set selection method. Section 3 introduces the five reduction rules. Section 4 describes how to reuse previously selected path sets to reduce the computation. Section 5 defines the enhanced zero-one optimal path set selection method. Section 6 proposes an enhanced approach to solving infeasible path problem. Section 7 evaluates the enhanced zero-one method, the original zero-one method, and a greedy method. Conclusion is given in Section 8. An algorithm of the five reduction rules is stated in the Appendix.

2. ZERO-ONE OPTIMAL PATH SET SELECTION METHOD (WANG ET AL., 1989) (LIN AND CHUNG, 1989)

2.1 Basic Concept

At first, the all-branches coverage criterion is used as the required coverage criterion to illustrate the concept of the zero-one optimal path set selection method. In discussion of structural testing, the structure of the program under test is usually mapped to a program digraph $G = (N, B)$, where N and B represent node set and branch set, respectively. Without loss of generality, it is assumed a sole source node and a sole terminal node both exist in the digraph. A path, starting from the source node and ending with the terminal node, is a sequence of nodes each connected by branches. Using network methodologies such as node-reduction (Beizer, 1984) or linearly independent circuits (McCabe, 1976), the complete path set P, which is defined as the set containing all paths, of program digraph G can then be constructed. A program with loops may lead to an extremely large number of paths. To alleviate this problem, it is considered sufficient in practice to limit loop iterations up to a constant number. The relationship between P and B can be represented by a branch-path coverage frequency matrix in which the cross entry of the ith row and the jth column represents the coverage frequency of path p_i over branch b_j. For example, consider the program digraph in Figure 1, the corresponding branch-path coverage frequency matrix is in Figure 2.

For the all-branches coverage criterion, the optimal path set selection problem can be defined as follows. In the complete path set P, which paths should be selected to guarantee that each branch in B is covered at least once and the number of selected paths is minimal? This is a decision problem. In the above example, define the variables x_i, $i \in \{1, 2, \ldots, 14\}$, let x_i be a decision variable corresponding to path p_i, and $x_i = 1$, if p_i is selected; 0, otherwise. Thus, the optimal path set selection problem can be formulated as:

$$\min \quad Z = \sum_{i=1}^{14} x_i$$

s.t.

$$\begin{bmatrix}
0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&1&1&0&1&0&0&1&0&0&0&0&0&0&0&0&0\\
1&0&0&1&1&1&0&1&0&1&0&0&0&0&0&0&0&0&0\\
1&0&0&1&0&1&0&0&1&0&0&0&0&0&1&0&1&0&1\\
1&0&0&1&0&0&1&0&0&0&1&0&1&0&1&0&1&0&1\\
1&0&0&1&0&1&0&1&0&0&1&0&1&0&1&0&1&0&1\\
1&0&0&1&0&0&1&0&0&0&0&1&0&1&1&0&1&0&1\\
1&0&0&1&0&1&0&1&0&0&0&1&0&1&1&0&1&0&1\\
1&0&0&1&0&1&0&0&1&0&0&0&0&0&1&1&0&1&1\\
1&0&0&1&0&0&1&0&0&0&1&0&1&0&1&1&0&1&1\\
1&0&0&1&0&1&0&1&0&0&1&0&1&0&1&1&0&1&1\\
1&0&0&1&0&0&1&0&0&0&0&1&0&1&1&1&0&1&1\\
1&0&0&1&0&1&0&1&0&0&0&1&0&1&1&1&0&1&1
\end{bmatrix}^T
*
\begin{bmatrix}
x_1\\x_2\\x_3\\x_4\\x_5\\x_6\\x_7\\x_8\\x_9\\x_{10}\\x_{11}\\x_{12}\\x_{13}\\x_{14}
\end{bmatrix}
\geq
\begin{bmatrix}
1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1
\end{bmatrix}.$$

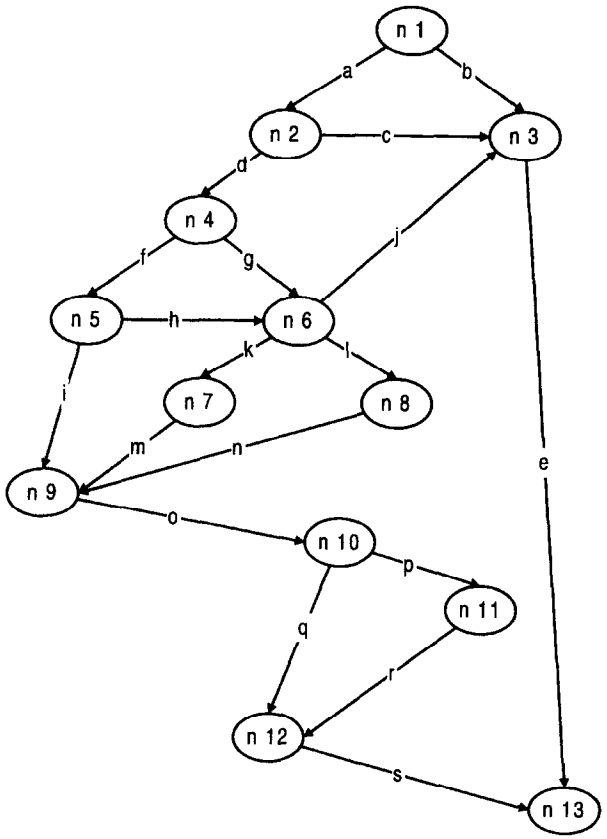

Figure 1. An example control flow graph.

The $Z = \sum_{i=1}^{14} x_i$ represents the number of selected paths and the constraint represents the total path coverage frequency over each branch is no less than one. In this way, the optimal path set selection problem is formulated into a zero-one integer programming problem.

In a generalized form, consider the program digraph $G = (N, B)$ and its complete path set $P = \{p_1, p_2, \ldots, p_m\}$, define a decision variable array $X(m \times 1) = [(x_i)]$, $i \in \{1, 2, \ldots, m\}$, and let x_i be a decision variable corresponding to path p_i. The optimal path set selection problem is formulated into a zero-one integer programming model as follows:

$$\min \quad Z = \sum_{i=1}^{m} x_i$$

$$\text{s.t.} \quad \sum_{i=1}^{m} f_{ij} x_i \geq 1, \qquad \forall j \in \{1, 2, \ldots, |B|\},$$

$$x_i = 0 \text{ or } 1, \qquad i \in \{1, 2, \ldots, m\},$$

where f_{ij} represents the coverage frequency of path p_i over branch b_j, $|B|$ denotes the number of total branches, and $\sum_{i=1}^{m} f_{ij} x_i$ represents the total path coverage frequency over branch b_j. The model can be expressed into matrix form as follows:

$$\min \quad Z = 1^T X,$$

$$\text{s.t.} \quad F^T X \geq 1, \qquad x_i = 0 \text{ or } 1, \qquad i \in \{1, 2, \ldots, m\},$$

where $1 = [1\,1 \ldots 1]^T$, $X(m \times 1) = [x_1 x_2 \ldots x_m]^T$,

Path \ Branch	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s
p_1=be	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p_2=ace	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p_3=adgje	1	0	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0
p_4=adfhje	1	0	0	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0
p_5=adfioqs	1	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0	1	0	1
p_6=adgkmoq	1	0	0	1	0	0	1	0	0	0	1	0	1	0	1	0	1	0	1
p_7=adfhkmoqs	1	0	0	1	0	1	0	1	0	0	1	0	1	0	1	0	1	0	1
p_8=adglnoqs	1	0	0	1	0	0	1	0	0	0	0	1	0	1	1	0	1	0	1
p_9=adfhlnoqs	1	0	0	1	0	1	0	1	0	0	0	1	0	1	1	0	1	0	1
p_{10}=adfioprs	1	0	0	1	0	1	0	0	1	0	0	0	0	0	1	1	0	1	1
p_{11}=adgkmoprs	1	0	0	1	0	0	1	0	0	0	1	0	1	0	1	1	0	1	1
p_{12}=adfhkmoprs	1	0	0	1	0	1	0	1	0	0	1	0	1	0	1	1	0	1	1
p_{13}=adglnoprs	1	0	0	1	0	0	1	0	0	0	0	1	0	1	1	1	0	1	1
p_{14}=adfhlnoprs	1	0	0	1	0	1	0	1	0	0	0	1	0	1	1	1	0	1	1

Figure 2. Branch-path coverage frequency matrix.

and

$$F(m \times |B|) = [(f_{ij})] = \begin{bmatrix} f_{11} & \cdot & \cdot & \cdot & f_{1|B|} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ f_{i1} & \cdot & f_{ij} & \cdot & f_{i|B|} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ f_{m1} & \cdot & \cdot & \cdot & f_{m|B|} \end{bmatrix},$$

F is referred to as coverage frequency matrix.

2.2 Generalization of the Model

Consider the facts that (1) different paths may have different test costs, and (2) only critical branches need to be tested sometimes. Thus, this model is generalized as follows: Define a weight matrix $W(m \times n) = [(w_{ij})]$ and a cost array $C(n \times 1) = [(c_i)]$, where $0 < c_i < \infty$. The objective function is generalized as $Z = C^T W^T X$. Define a coverage requirement array $R(n \times 1) = [(r_i)]$, where r_i represents the coverage requirement for branch i, $r_i = 0$ or 1. The constraint inequation is generalized as $F^T X \geq R$. Thus, the optimal path set selection model is generalized as follows:

$$\min \quad Z = C^T W^T X,$$
$$\text{s.t.} \quad F^T X \geq R,$$
$$X = [(x_i)], \qquad x_i = 0 \text{ or } 1, \qquad i \in \{1, 2, \ldots, m\}.$$

In this generalized model, the variable arrays and matrices such as C, W, F, and R can be assigned specified optimization criteria and test requirements. For example, consider the all-nodes coverage criterion. Define node-path coverage frequency matrix $F_n(m \times n) = [(f_{ij})]$, where f_{ij} stands for coverage frequency of path p_i over node n_j. The constraint inequation thus becomes $F_n^T X \geq 1$ for the all-nodes coverage criterion. How to apply the generalized constraint inequation to other coverage criteria and how to assign specified optimization criteria are described in detail in (Wang et al., 1989) (Lin and Chung, 1989).

The model definitely has a solution because the program under test is assumed well-formed and $X(m \times 1) = [1\,1 \ldots 1]^T$ is a solution in the worst case. Finding a solution of the model is straightforward and there are two ways to solve it. One is exhaustive enumeration which lists all $(2^m - 1)$ combinations to check and finds an entire set of solutions. The other, finding only one solution, utilizes the branch-and-bound approach and many effective algorithms are available (Hillier and Lieberman, 1980) (Syslo et al., 1983). Among them, the Balas' zero-one additive algorithm is considered to be the fastest. The complexity of the zero-one integer programming method is proportional to $(2^{|Path|} \times |Component|)$ where $|Path|$ represents the number of candidate paths while $|Component|$ represents the number of components to be covered.

The major drawback of the zero-one integer programming method is that the computation time is too long because it is proportional to $(2^{|Path|} \times$

|*Component*|). Obviously, the computation time can be reduced if the size of the coverage frequency matrix (i.e., the number of rows and the number of columns) is reduced. The deletion of a column corresponds to the decrease of a constraint in the formulation. A column can be deleted if and only if its corresponding constraint is redundant. On the other hand, the deletion of a row corresponds to the decrease of a decision variable in the formulation. A decision variable can be deleted if and only if the value of the decision variable can be predetermined. In the next section, five reduction rules are developed by observing the relationship between candidate paths and the components to be covered.

3. FIVE RULES TO REDUCE THE COMPUTATION

It is easy to observe that:

- if a component is not required to be covered, the component can be ignored in the path selection problem,
- if a component is required to be covered and is covered by only one path, the path which covers the component must be selected,
- if any path covering a component, say c_i, also covers another component, say c_j, then the requirement that c_i and c_j must be covered at least once can be reduced to c_i must be covered at least once, and
- if a path, say p_i, covers all components covered by another path, say p_j, and some additional components, then p_j can be ignored in the path selection problem owing to the existence of p_i.

Based on these observations, this paper proposes five reduction rules. To illustrate the five rules formally, the following notation are defined:

Symbol	Representation
$F(m \times n) = [(f_{ij})]$	coverage frequency matrix,
$R(n \times 1) = [(r_i)]$	coverage requirement array, $r_i = 0$ or 1,
Row_i	the ith row matrix of F,
Col_j	the jth column matrix of F,
$\|Row_i\| = \sum_{k=1}^{n} f_{ik}$	the summation of the ith row's non-zero entries,
$\|Col_j\| = \sum_{k=1}^{m} f_{kj}$	the summation of the jth column's non-zero entries.

Rule One (Surely Satisfied Constraint). If a component is not required to be covered, its corresponding constraint is surely satisfied and thus can be ignored.

Rule 1 *If $r_i = 0$, $i \in \{1, 2, \ldots, n\}$, then (a) delete Col_i, (b) delete the ith row of R (i.e., r_i).*

Proof. Since $r_i = 0$, the corresponding constraint of Col_i is $\sum_{k=1}^{m} f_{ki} x_k \geq 0$. Due to $f_{ki} \geq 0$, and $x_k = 0$ or 1, $k \in \{1, 2, \ldots, m\}$, the above constraint is surely satisfied and thus can be ignored.

Rule Two (Essential Path). A path is essential if and only if some component is covered by the path only. For example, assuming that the all-branches coverage criterion is required, the path $p_2 = (ace)$ in the control flow graph shown in Figure 1 must be selected because it is the only one passing through branch c. In addition, after p_2 is selected, the branches a, c, and e can be ignored in the latter computation because they have been covered by the selected path p_2.

Rule 2 *If $|Col_j| = f_{kj}$ and $f_{kj} \geq 1$, $k \in \{1, 2, \ldots, m\}$, $j \in \{1, 2, \ldots, n\}$, then (a) set $x_k = 1$, (b) delete Col_i and corresponding r_i, $\forall f_{ki} \geq 1$, $i \in \{1, 2, \ldots, n\}$, and (c) delete Row_k.*

Proof.

(1) Since the jth component is covered by path p_k only, the jth constraint $f_{kj} x_k \geq 1$ can be satisfied only when $x_k = 1$.

(2) Let f_{ki} be any non-zero entry of Row_k, $i \in \{1, 2, \ldots, n\}$, its corresponding constraint is:

$$\sum_{r=1}^{m} f_{ri} x_r = f_{1i} x_1 + \cdots + f_{ki} x_k + \cdots + f_{mi} x_m \geq 1.$$

Since $f_{ki} \geq 1$, if $x_k = 1$, the constraint is satisfied. Let F' represent the matrix obtained from F after deleting the corresponding columns of f_{ki}, $\forall f_{ki} \geq 1$, $i \in [1, 2, \ldots, n\}$. Any solution of $(F')^T X \geq 1$ with $x_k = 1$ also satisfies $F^T X \geq 1$. This implies that if $x_k = 1$ is predetermined, the constraint is redundant and thus its corresponding Col_i can be deleted.

(3) Let $F'(m \times n')$ represent the matrix obtained from F after deleting the corresponding columns of f_{ki}, $\forall f_{ki} \geq 1$, $i \in \{1, 2, \ldots, n\}$. The constraint inequation of $(F')^T X \geq 1$ is

$$\sum_{r=1}^{m} f_{rj} x_r = f_{1j} x_1 + \cdots + f_{mj} x_m \geq 1, \forall j \in \{1, 2, \ldots, n'\}.$$

Since $|Row_k| = 0$, i.e., $f_{kj} = 0, \forall j \in \{1, 2, \ldots, n'\}$,

the above expression is equivalent to:

$$\sum_{r=1}^{m} f_{rj}x_r = f_{1j}x_1 + \cdots + f_{(k-1)j}x_{(k-1)} + f_{(k+1)j}x_{(k+1)} \cdots + f_{mj}x_m \geq 1, \quad \forall j \in \{1,2,\ldots,n'\}.$$

Thus, Row_k can be deleted.

Rule Three (Dominating Component / Dominating Column). A component, say c_i, dominates another component, say c_j, if and only if any path covering c_i also covers c_j. For example, in Fig. 1, the branch f dominates branch d because any path covering f also covers d. If f has been covered by some selected path, d is covered too. Thus the requirement that d must be covered at least once can be ignored if f is required to be covered at least once.

Rule 3 *If* $|Col_i| > 0$, $|Col_j| > 0$, *and* $f_{kj} \geq f_{ki}$, $\forall k \in \{1,2,\ldots,m\}$, *then delete* Col_j, *delete* r_j.

Proof. The constraint of Col_i is:

$$\sum_{k=1}^{m} f_{ki}x_k \geq 1 \qquad \text{Constraint(1)}$$

The constraint of Col_j is:

$$\sum_{k=1}^{m} f_{kj}x_k \geq 1 \qquad \text{Constraint(2)}$$

Since $f_{kj} - f_{ki} \geq 0$, $\forall k \in \{1,2,\ldots,m\}$ and $x_i = 0$ or 1, $i \in \{1,2,\ldots,m\}$, $\sum_{k=1}^{m}(f_{kj} - f_{ki})x_k \geq 0$. Since $\sum_{k=1}^{m}(f_{kj} - f_{ki})x_k = \sum_{k=1}^{m} f_{kj}x_k - \sum_{k=1}^{m} f_{ki}x_k$, $\sum_{k=1}^{m} f_{kj}x_k \geq \sum_{k=1}^{m} f_{ki}x_k$ Inequation (1)

Inequation (1) implies that any solution satisfying Constraint (1) also satisfies Constraint (2). That is, Constraint (2) is redundant and thus can be deleted.

Note that, once reduction rule 3 is applied, the remaining components are equal to the "unconstrained duas" mentioned in (Marre and Bertolino, 1996) if the required coverage criterion is all-uses and are also equal to the "essential branches" mentioned in (Chusho, 1987) if the required coverage criterion is all-branches. The difference is that the proposed technique is based on the relation between path and components while the others are based on graph theory.

Rule Four (Dominating Path / Dominating Row). Path p_i dominates path p_j if and only if p_i covers both all p_j's components and some additional components. In this situation, p_j can be ignored owing to the existence of p_i. For example, in the program digraph shown in Figure 3, path $p_2 = (abcdbce)$ dominates $p_1 = (abce)$ because p_2 covers both the routine of p_1 and an additional loop (dbc). Thus p_1 can be ignored in the path selection problem because the path p_2 is involved in the consideration.

Rule 4 *If* $f_{ik} \geq f_{jk}$, $\forall k \in \{1,2,\ldots,n\}$, *then* (*a*) *set* $x_j = 0$, (*b*) *delete* Row_j.

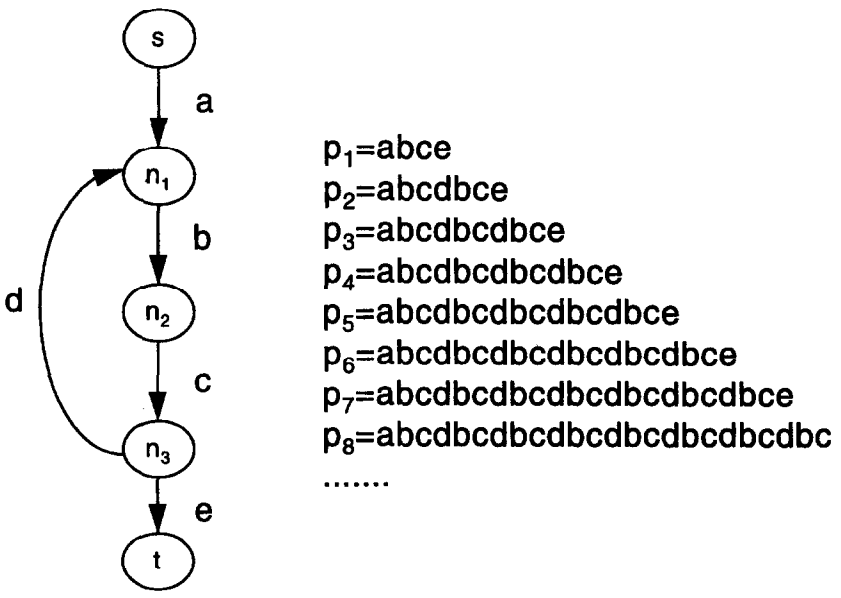

Figure 3. A control flow graph and path list.

Proof. Assume that $X = (x_1, x_2, \ldots, x_m)^T$ with $x_j = 1$ is a feasible solution of $F^T X \geq 1$. If X is modified to be X' with $x_j = 0$ and $x_i = 1$, the X' also satisfies $F^T X \geq 1$ because the components covered by X are also covered by X'. That is, X' is also a feasible solution. This means that we can set $x_j = 0$ and delete Row_j from the matrix without affecting the solution of the problem.

Rule Five (Zero Row). Path p_i is a zero path if and only if it does not cover any component. This situation happens after Rule 2 is applied while the essential path also dominates another path. In this case, the zero path can be directly deleted without affecting the problem solution.

Rule 5 *If* $f_{ij} = 0$, $\forall j \in \{1, 2, \ldots, n\}$, $i \in \{1, 2, \ldots, m\}$, *then* (*a*) *delete* Row_i, (*b*) *set* $x_i = 0$.

Proof.

(1) The constraint inequation of $F^T X \geq 1$ is

$$\sum_{k=1}^{m} f_{kj} x_k \geq 1, \qquad \forall j \in \{1, 2, \ldots, n\}.$$

Since $|Row_i| = 0$, i.e., $f_{ij} = 0$, $\forall j \in \{1, 2, \ldots, n\}$, the above inequation is equivalent to:

$$\sum_{k=1}^{m} f_{kj} x_k = f_{1j} x_1 + \cdots + f_{(i-1)j} x_{(i-1)} + f_{(i+1)j} x_{(i+1)} \cdots + f_{mj} x_m \geq 1, \qquad \forall j \in \{1, 2, \ldots, n\}.$$

Thus, Row_i can be deleted.

(2) Both $x_i = 0$ and $x_i = 1$ satisfy all constraints of $F^T X \geq 1$. However, $x_i = 0$ must be chosen to obtain a minimal number of selected paths.

Among the five reduction rules, rule one should be applied first, and then the other four rules should be applied repeatedly until none of the rules can be activated. A formal algorithm is stated in the Appendix to demonstrate how to apply the reduction rules. Rule 1 is not included in the algorithm because it is very simple and should be applied in advance.

In order to show how effective the five reduction rules are, an example is given below. A software package LINDO (Linear INteractive Discrete Optimizer) (Schrage, 1987), which applies Balas' algorithm, is used to solve the formulated zero-one integer programming problems. Consider the control flow graph shown in Figure 1, assuming that the all-branches coverage criterion is required. Based on the branch-path coverage frequency matrix shown in Figure 2, the optimal path set selection problem is formulated in LINDO as follows:

```
MIN  X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8
     +X9 + X10 + X11 + X12 + X13 + X14
SUBJECT TO
X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 + X10
     +X11 + X12 + X13 + X14 > = 1
X1 > = 1
X2 > = 1

X3 + X4 + X5 + X6 + X7 + X8 + X9 + X10 + X11
     +X12 + X13 + X14 > = 1
X1 + X2 + X3 + X4 > = 1
X4 + X5 + X7 + X9 + X10 + X12 + X14 > = 1
X3 + X6 + X8 + X11 + X13 > = 1
X4 + X7 + X9 + X12 + X14 > = 1
X5 + X10 > = 1
X3 + X4 > = 1
X6 + X7 + X11 + X12 > = 1
X8 + X9 + X13 + X14 > = 1
X6 + X7 + X11 + X12 > = 1
X8 + X9 + X13 + X14 > = 1
X5 + X6 + X7 + X8 + X9 + X10 + X11 + X12
     +X13 + X14 > = 1
X10 + X11 + X12 + X13 + X14 > = 1
X5 + X6 + X7 + X8 + X9 > = 1
X10 + X11 + X12 + X13 + X14 > = 1
X5 + X6 + X7 + X8 + X9 + X10 + X11 + X12
     +X13 + X14 > = 1.
```

After 12 iteration steps, LINDO renders the following solution:

OBJECTIVE FUNCTION VALUE 6

VARIABLE	*VALUE*	*VARIABLE*	*VALUE*
X1	1.000000	X8	.000000
X2	1.000000	X9	.000000
X3	.000000	X10	.000000
X4	1.000000	X11	.000000
X5	1.000000	X12	.000000
X6	1.000000	X13	1.000000
X7	.000000	X14	.000000

The paths whose corresponding decision variable's value = 1.000000 are selected. That is p_1, p_2, p_4, p_5, p_6, p_{13} are selected in the optimal path set.

Now, the five reduction rules are applied. In this example, since branch b is uniquely covered by p_1 and branch c is uniquely covered by p_2, by rule 2, p_1 and p_2 are essential paths and thus must be selected. After setting $x_1 = 1$, $x_2 = 1$, deleting the columns whose corresponding branches (a, b, c, e) are covered by p_1 and p_2, and the rows corresponding to p_1 and p_2, the reduced coverage frequency

matrix is as follows:

Path \ Branch	d	f	g	h	i	j	k	l	m	n	o	p	q	r	s
p_3 = adgje	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0
p_4 = adfhje	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0
p_5 = adfioqs	1	1	0	0	1	0	0	0	0	0	1	0	1	0	1
p_6 = adgkmoq	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1
p_7 = adfhkmoqs	1	1	0	1	0	0	1	0	1	0	1	0	1	0	1
p_8 = adglnoqs	1	0	1	0	0	0	0	1	0	1	1	0	1	0	1
p_9 = adfhlnoqs	1	1	0	1	0	0	0	1	0	1	1	0	1	0	1
p_{10} = adfioprs	1	1	0	0	1	0	0	0	0	0	1	1	0	1	1
p_{11} = adgkmoprs	1	0	1	0	0	0	1	0	1	0	1	1	0	1	1
p_{12} = adfhkmoprs	1	1	0	1	0	0	1	0	1	0	1	1	0	1	1
p_{13} = adglnoprs	1	0	1	0	0	0	0	1	0	1	1	1	0	1	1
p_{14} = adfhlnoprs	1	1	0	1	0	0	0	1	0	1	1	1	0	1	1

By rule 3, since branches d, f, m, n, o, r, s are dominated by branches g, h, k, l, k, p, q, respectively, the columns of branches d, f, m, n, o, r, s can be deleted. The further reduced matrix is as follows:

Path \ Branch	g	h	i	j	k	l	p	q
p_3 = adgje	1	0	0	1	0	0	0	0
p_4 = adfhje	0	1	0	1	0	0	0	0
p_5 = adfioqs	0	0	1	0	0	0	0	1
p_6 = adgkmoq	1	0	0	0	1	0	0	1
p_7 = adfhkmoqs	0	1	0	0	1	0	0	1
p_8 = adglnoqs	1	0	0	0	0	1	0	1
p_9 = adfhlnoqs	0	1	0	0	0	1	0	1
p_{10} = adfioprs	0	0	1	0	0	0	1	0
p_{11} = adgkmoprs	1	0	0	0	1	0	1	0
p_{12} = adfhkmoprs	0	1	0	0	1	0	1	0
p_{13} = adglnoprs	1	0	0	0	0	1	1	0
p_{14} = adfhlnoprs	0	1	0	0	0	1	1	0

Since no more reduction rule can be applied, LINDO is used again but only the reduced matrix is considered. After 9 iteration steps, LINDO renders *OBJECTIVE FUNCTION VALUE* 4, with p_3, p_7, p_{10}, and p_{14} selected. Together with pre-selected p_1 and p_2, the total number of selected paths is six.

Note that before and after the reduction rules are applied, the value of the objective function is the same. However, the size of the coverage frequency matrix is reduced from 14×19 to 12×8, and the number of iteration steps in LINDO is decreased from 12 to 9.

4. REUSE OF PREVIOUSLY SELECTED PATH SETS

In structural program testing, the steps are:

1. select a coverage criterion,
2. find a path set satisfying the selected coverage criterion,
3. generate a corresponding test case for each selected path,
4. exercise each test case and check whether the corresponding path is executed and whether the output is the same as the expected,
5. if errors are detected, make an error report (suppose that the tester is not the debugger),
6. if the coverage criterion is deemed sufficient, stop the testing; otherwise, select a strong coverage criterion and then repeat steps 2–6.

A coverage criterion, say C_2, is stronger than (subsumes) another coverage criterion, say C_1, if and only if any path set satisfying C_2 also satisfies C_1. Many testing criteria have been compared using the subsumption relation (Rapps and Weyuker, 1985).

Assume that C_2 is the next selected coverage criterion while P_1 is an optimal path set for the previously selected coverage criterion C_1. To find an optimal path set for C_2, a new coverage frequency

matrix is built, zero-one integer programming model is formulated, and then LINDO is used to obtain the solution. Assuming that the reduction rules are not used, the computation for C_2 is longer than that for C_1 because the stronger the coverage criterion is, the larger the number of components to be covered is. For example, consider the program digraph shown in Figure 1, the number of nodes is 13 (see Figure 4) while that of branches is 19 (see Figure 2).

To decrease the long computation for C_2, the size of the corresponding coverage frequency matrix must be reduced. In addition to the five reduction rules, previously selected path sets should also be reused. To reuse previously selected path sets, the relationship between different coverage criteria and corresponding coverage frequency matrices should be studied. Different coverage criteria are focused on different types of program components, but no matter what coverage criterion is required, the corresponding optimal path set is selected from the same complete path set. Thus, a program has different coverage frequency matrices for different coverage criteria with the same number and elements of rows (candidate paths) but different number and elements of columns (components to be covered).

Since the optimal path sets are selected from the same complete path set, some previously selected and tested paths may also appear in the optimal path set for the next coverage criterion. The optimal path set of the next coverage criterion with maximal number of previously selected paths is desired because minimal number of new paths are to be tested and thus minimal number of new test cases are to be generated. The best case is to reuse all previously selected paths and select an optimal path set from the remaining paths to cover remaining components. In this way, the information of previously selected path sets can be reused to reduce the size of coverage frequency matrix, and thus the computation.

Assume that the previously selected path set is P_1 while the coverage frequency matrix corresponding to the next coverage criterion is F. The rows of F can be partitioned into two groups: *the paths in* P_1, and *the paths not in* P_1. The components to be covered can also be partitioned into two groups: *the components covered by* P_1 and *the components not covered by* P_1. Thus, F can be reorganized as:

Path \ *Component*	*Components not covered by* P_1	*Components covered by* P_1
Paths in P_1	*All-Zero*	$F_{12}(i, j)$
Paths not in P_1	$F_{21}(i, j)$	$F_{22}(i, j)$

Since only the components not covered by the paths in P_1 must be covered by the newly added paths, the zero-one optimal path set selection method should be applied to the reduced coverage

Path \Node	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}	n_{11}	n_{12}	n_{13}
p_1=be	1	0	1	0	0	0	0	0	0	0	0	0	1
p_2=ace	1	1	1	0	0	0	0	0	0	0	0	0	1
p_3=adgje	1	1	1	1	0	1	0	0	0	0	0	0	1
p_4=adfhje	1	1	1	1	1	1	0	0	0	0	0	0	1
p_5=adfioqs	1	1	0	1	1	0	0	0	1	1	0	1	1
p_6=adgkmoq	1	1	0	1	0	1	1	0	1	1	0	1	1
p_7=adfhkmoqs	1	1	0	1	1	1	1	0	1	1	0	1	1
p_8=adglnoqs	1	1	0	1	0	1	0	1	1	1	0	1	1
p_9=adfhlnoqs	1	1	0	1	1	1	0	1	1	1	0	1	1
p_{10}=adfioprs	1	1	0	1	1	0	0	0	1	1	1	1	1
p_{11}=adgkmoprs	1	1	0	1	0	1	1	0	1	1	1	1	1
p_{12}=adfhkmoprs	1	1	0	1	1	1	1	0	1	1	1	1	1
p_{13}=adglnoprs	1	1	0	1	0	1	0	1	1	1	1	1	1
p_{14}=adfhlnoprs	1	1	0	1	1	1	0	1	1	1	1	1	1

Figure 4. Node-path coverage frequency matrix.

Path \ Branch	b	c	g	i	q
p_1=be	1	0	0	0	0
p_2=ace	0	1	0	0	0
p_3=adgje	0	0	1	0	0
p_5=adfioqs	0	0	0	1	1
p_6=adgkmoqs	0	0	1	0	1
p_7=adfhkmoqs	0	0	0	0	1
p_8=adglnoqs	0	0	1	0	1
p_9=adfhlnoqs	0	0	0	0	1
p_{10}=adfioprs	0	0	0	1	0
p_{11}=adgkmoprs	0	0	1	0	0
p_{13}=adglnoprs	0	0	1	0	0

Figure 5. Reduced branch-path coverage frequency matrix.

frequency matrix $F_{21}(i, j)$ instead of F. The computation to find an optimal path set for $F_{21}(i, j)$ is certainly less than that for F. Consider the control flow graph shown in Figure 1, assuming that the previously selected optimal path set for the all-nodes coverage criterion is $P_1 = \{p_4, p_{12}, p_{14}\}$, after deleting the rows corresponding to the paths in P_1 and the branches covered by the paths in P_1, the size of the branch-path coverage frequency matrix is reduced from a 14×19 matrix (see Figure 2) to a 11×5 matrix (see Figure 5). Since the obtained optimal path set covers all components not covered by P_1, the union of the obtained path set and P_1 satisfies the all-branches coverage criterion.

The minimal requirement for the obtained path set is that the number of paths in it is no more than the number of newly selected paths in any optimal path set found by existing optimal path set selection methods; otherwise, the tester must spend more time to generate corresponding test cases. The satisfaction of this requirement can be proven as follows: Let

- P be the complete path set,
- C_1 be the previously satisfied coverage criterion,
- P_1 be the previously selected optimal path set satisfying C_1,
- C_2 be the next selected coverage criterion,
- F be the coverage frequency matrix for C_2,
- Y be any optimal path set satisfying C_2,
- F' be the reduced matrix of F derived by deleting the rows whose corresponding paths are in P_1 and the columns whose corresponding components are covered by the paths in P_1,
- X be an optimal path set found by applying the zero-one integer programming method to F',
- $A - B$ represent the path set which contains the paths in path set A but not also in path set B,
- $|path\text{-}set|$ represent the number of paths in a given path set.

Show that $|X| \leq |Y - P_1|$.

Proof. F can be re-organized as follows:

Path \ Component	*Components not covered by P_1*	*Components covered by P_1*
P_1	*All Zero*	$F_{12}(i, j)$
$P - P_1$	$F_{21}(i, j)$	$F_{22}(i, j)$

Y can be partitioned into: *paths also in* P_1 (i.e., $Y \cap P_1$) and *paths not also in* P_1 (i.e., $Y - P_1$). The components covered by P_1 can be partitioned into: *covered by* Y and *not covered by* Y. Thus, F can be further re-organized as follows:

Path \ Component		*Components not covered by P_1*	*Components Covered by P_1*
			Not Covered by $Y \cap P_1$
P_1		*All Zero*	$F_{12}(i, j)$
$P - P_1$	$Y - P_1$	$F_{31}(i, j)$	$F_{32}(i, j)$
		$F_{41}(i, j)$	$F_{42}(i, j)$

If we delete the $Y \cap P_1$ row (i.e., the shadowed row) and the components covered by $Y \cap P_1$ (i.e., the shadowed column), the reduced coverage frequency matrix is:

Path \ *Component*		*Components not covered by* P_1	*Components Covered by* P_1 *but Not Covered by* $Y \cap P_1$
P_1		*All Zero*	$F_{12}(i, j)$
$P - P_1$	$Y - P_1$	$F_{31}(i, j)$	$F_{32}(i, j)$
		$F_{41}(i, j)$	$F_{42}(i, j)$

The path set $Y - P_1$ can be treated as a path set selected from the $P - P_1$ which covers the components listed in the reduced coverage frequency matrix.

The path set X is an optimal path set satisfying the following reduced coverage frequency matrix:

Path \ *Component*		*Components not covered by* P_1
$P - P_1$	$Y - P_1$	$F_{31}(i, j)$
		$F_{41}(i, j)$

Since the components covered by X are only a subset of the components covered by $Y - P_1$. If $|Y - P_1| < |X|$, it means that there exists a path set $Y - P_1$, which is also selected from $P - P_1$, covering all the components covered by X but the number of contained paths is less than X. This contradicts with the fact that X is an "optimal" (remember that optimal means minimal) path set. So, $|X| \leq |Y - P_1|$.

The meaning of $|X| \leq |Y - P_1|$ is not only that we can efficiently find a path set to satisfy a stronger coverage criterion but also the number of the needed test cases is reduced, although the path set $P_1 + X$ is not always an optimal path set for the next coverage criterion.

5. ENHANCED ZERO-ONE OPTIMAL PATH SET SELECTION METHOD

Adding the five reduction rules and reusing previously selected path sets, the original zero-one optimal path set selection method is enhanced as follows:

For the first selected coverage criterion: (usually the all-statements coverage criterion)

Step 1: generate a corresponding coverage frequency matrix,

Step 2: apply the five reduction rules to reduce the matrix size,

Step 3: translate the reduced coverage frequency matrix into constraint inequalities (each column corresponds to a constraint inequation), and translate the goal that the number of selected paths is minimal into objective function, and

Step 4: solve the formulated zero-one integer programming model by available package (for example LINDO).

For the next selected stronger coverage criterion, add the following step between step 1 and step 2:

- Eliminate the rows corresponding to the paths in the previously selected path sets and the columns corresponding to the components covered by the paths in the previously selected path sets.

6. INFEASIBLE PATH CONSIDERATION

As in practice, the program digraph rather than the program itself is referred in the discussion of structural testing. However, as is pointed out by (Ntafos and Kakimi, 1979), infeasible paths may exist. Inclusion of infeasible test paths is meaningless. Identification of infeasible paths is an undecidable problem and is still under study (Gabow et al., 1976) (Hedley and Hennell, 1985).

An intuitive approach is deleting the infeasible paths from the complete path set, regenerating the coverage frequency matrix, reformulating the zero-one integer programming model, and then recomputing to select another optimal path set (Lin and Chung, 1989). The drawback is that once an infeasible path is found, the whole process is repeated with almost the same complexity (since only the infeasible paths are deleted) and computation time.

Similar to the idea of reusing previously selected path sets mentioned in Section 4, the feasible paths in the optimal path set, which contains infeasible paths, can be reused. That is, we just have to find some new test paths from the remaining unselected paths to cover the components that are covered by the infeasible test paths only. An enhanced approach is proposed as follows:

1. delete the rows corresponding to infeasible paths that are in the optimal path set from the coverage frequency matrix,
2. delete the rows corresponding to feasible paths that are in the optimal path set and the columns corresponding to the components covered by feasible paths that are in the optimal path set from the coverage frequency matrix,

3. if any column becomes empty (i.e., all entries in the column are zero), report the corresponding component as a "candidate" of infeasible component and then delete the empty column,
4. find a new optimal path set from the reduced coverage frequency matrix,
5. repeat this procedure until the new optimal path set does not contain any infeasible path.

Step 3 takes infeasible components into consideration. It is sometimes quite difficult for the tester to identify them because there may be a large number of paths covering a component and a component is infeasible if and only if all of the associated paths are infeasible. Since the coverage frequency matrix does not necessarily represent all paths (as mentioned in section 2.1, loop iterations are limited up to a constant number), the component whose corresponding column is zero (i.e., all entries in the column are zero) is only a candidate of infeasible component. Intuitively, if no iteration instruction is involved or the tester inputs the correct iteration count, the matrix can be used to identify infeasible components while the tester has identified all the infeasible paths. Note that, once a candidate of infeasible component has been reported, the optimal path set obtained does not cover the corresponding component.

The final test path set is the union of the feasible paths in the optimal path sets that have been selected. Of course, if no infeasible path has ever been included, the final test path set is optimal. Even after the enhanced approach is applied, the final test path set is still nearly optimal. To be more clear, a special case is given below to show that once the enhanced approach is applied, the final test path set is not guaranteed to be optimal. Assume that the original coverage frequency matrix is as follows:

	a	b	c	d
p_1	0	0	1	1
p_2	1	1	0	0
p_3	0	1	1	0
p_4	1	0	0	1

Assume that the originally selected optimal path set is $\{p_3, p_4\}$ and p_3 is infeasible. By the enhanced approach, the rows corresponding to p_3 and p_4 are deleted but only the columns corresponding to the components covered by p_4 are deleted from the coverage frequency matrix. The reduced coverage frequency matrix is as follows:

	b	c
p_1	0	1
p_2	1	0

Obviously, the new optimal path set is $\{p_1, p_2\}$. Together with the feasible path in the originally selected optimal path set, the final test path set is $\{p_1, p_2, p_4\}$. However, $\{p_1, p_2\}$ is a path set covering all components with only two paths. Thus, once the enhanced approach is applied, the final test path set is not guaranteed to be optimal. In general, however, the final test path set is always nearly optimal especially for the case that the originally selected optimal path set contains many test paths and only one or two are infeasible.

7. EVALUATION

Minimization is only worthwhile if the reduction in cost derived from reducing the path set compensates for the cost of doing the minimization. Otherwise, greedy methods, which obtain near optimal path set with less computation time, is more practical.

Four experiments have been executed to evaluate the original zero-one method, the enhanced zero-one method, and the greedy method proposed by (Hsu and Chung, 1992). The greedy method selects a path a time and the path to be selected is always the one covering most remaining uncovered components.

These four experiments first apply the three different methods to obtain an optimal path set for the all-nodes coverage criterion (the result is shown in Table 1) and then to obtain an optimal path set for the all-edges coverage criterion (the result is shown in Table 2).

In Table 1, consider the enhanced zero-one method, the five reduction rules are applied first and then LINDO is used if there are any components not covered yet. Except for the fourth experiment, once the five reduction rules are applied, the optimal path set will be obtained (i.e., LINDO is not used at all). For the original zero-one method, LINDO is used to obtain the optimal path set. The LINDO used in these four experiments is a student edition for the IBM PC which can only handle 200 variables (i.e., 200 paths), therefore it cannot handle the fourth experiment.

In Table 2, consider the enhanced zero-one method, the previously selected path set is reused to reduce the size of the coverage frequency matrix and then the five reduction rules are applied. In these four experiments, once the five reduction rules are

Table 1. Experiment Result for All-Nodes Coverage Criterion

	Enhanced Zero-One Method			Original Zero-One Method		Greedy Method	
All-Nodes Coverage	The time it takes to apply the five reduction rules	The time the optimization takes with the reduction	The size of the resulting path set	The time the optimization takes without the reduction	The size of the resulting path set	The time to derive the path set with greedy method	The size of path set derived by greedy method
Experiment 1 3 paths 8 nodes	182 micro second	No	2 paths {p1, p3}	5950 micro second	2 paths {p1, p3}	116 micro second	3 paths {p1, p2, p3}
Experiment 2 8 paths 9 nodes	452 micro second	No	2 paths {p5, p7}	7365 micro second	2 paths {p5, p8}	207 micro second	2 paths {p3, p5}
Experiment 3 14 paths 13 nodes	1112 micro second	No	3 paths {p4, p12, p14}	10908 micro second	3 paths {p1, p8, p12}	402 micro second	3 paths {p1, p8, p12}
Experiment 4 218 paths 28 nodes	60072 micro second	6008 micro second	6 paths {p173, p176, p191, p197, p203, p215}	LINDO Cannot handle!	No	9572 micro second	6 paths {p15, p21, p30, p33, p99, p155}

applied, the path set will be obtained (i.e., LINDO is not used at all). For the original zero-one method, LINDO is used to obtain the optimal path set. The student edition LINDO can not handle the fourth experiment because the number of paths is more than two hundred.

These four experiments are executed on an IBM compatible PC with Intel Pentium 100 CPU and 16 Mega RAM. The control flow graphs used in experiment 1, 2, 3 and 4 are shown in Figure 6, Figure 7, Figure 1, and Figure 8.

From the experiment result, it is obvious that the enhanced zero-one method does greatly enhance the original zero-one method by both reducing the computation time and the number of new paths to test. In addition, as shown in the fourth experiment, the enhanced zero-one method can handle the cases that can not be handled by the original zero-one method. Compared with the greedy method, the enhance zero-one method has the following two advantages: (1) For the first coverage criterion, the enhanced zero-one method guarantees the path set obtained is optimal while the greedy method does not, (2) For the second and subsequent coverage criterion, the enhanced zero-one method requires less computation time than the greedy method because previously selected path sets are reused to reduce the size of the corresponding coverage frequency matrix.

8. CONCLUSION

Usually, applying structural testing to a program is an iterative process. At first a weak coverage criterion is selected (e.g., all statements coverage), once it is satisfied, a stronger coverage criterion is required. For each coverage criterion, a path set is selected to satisfy the requirement and if the selected path set includes infeasible paths, another path set is selected again. Assuming that the cost of testing each path is the same, the path set satisfying the coverage criterion with minimal number of paths is desired. Finding a path set satisfying a required coverage criterion with minimal number of paths is referred to as "optimal path set selection problem". The zero-one optimal path set selection method is a generalized method that can not only be applied in structural testing to find an optimal path set satisfying the required coverage criterion but also in regression testing to find a minimal number of previously executed test cases to fully retest every affected program element at least once (Fischer, 1977) (Fischer et al., 1981) (Hartmann and Robson, 1990). The major drawback of this method is that for a large program the computation may take ten or more hours because the computation is exponentially proportional to the number of paths and proportional to the number of components. Due to the iterative nature of structural testing, the long computation becomes a serious problem that must be enhanced.

The enhanced zero-one optimal path set method can speed up the structural program testing process by: (1) For the first selected coverage criterion, the five reduction rules are applied ahead and then the zero-one optimal path set selection method is used to efficiently find an optimal path set. (2) For each next selected coverage criterion, the previously se-

Table 2. Experiment Result for All-Edges Coverage Criterion

	Enhanced Zero-One Method				Original Zero-One Method			Greedy Method		
All-Edges Coverage	The size of the matrix after reuse	The time it takes to apply the five reduction rules	The time the optimization takes with the reduction	The size of newly selected path	The time the optimization takes without the reduction	The size of the resulting path set	The size of newly selected path	The time to derive the path set with greedy method	The size of path set derived by greedy method	The size of newly selected path
Experiment 1 3 paths 9 branches	row = 1 col = 1	0 micro second	No	1 path {p2}	5909 micro second	3 paths {p1, p2, p3}	1 path {p2}	122 micro second	3 paths {p1, p2, p3}	0 path { }
Experiment 2 8 paths 11 branches	row = 6 col = 1	0 micro second	No	1 path {p2}	7829 micro second	2 paths {p4, p5}	1 path {p4}	242 micro second	2 paths {p4, p5}	1 path {p4}
Experiment 3 14 paths 19 branches	row = 11 col = 5	221 micro second	No	4 paths {p1, p2, p5, p6}	13 micro second	6 paths {p1, p2, p4, p5, p6, p13}	5 paths {p2, p4, p5, p6, p13}	452 micro second	6 paths {p1, p2, p3, p5, p8, p12}	3 paths {p2, p3, p5}
Experiment 4 218 paths 41 edges	row = 212 col = 4	4433 micro second	No	3 paths {p1, p6, p36}	LINDO Can not handle!	No	No	12102 micro second	8 paths {p1, p15, p21, p30, p33, p36, p99, p158}	3 paths {p1, p36, p158}

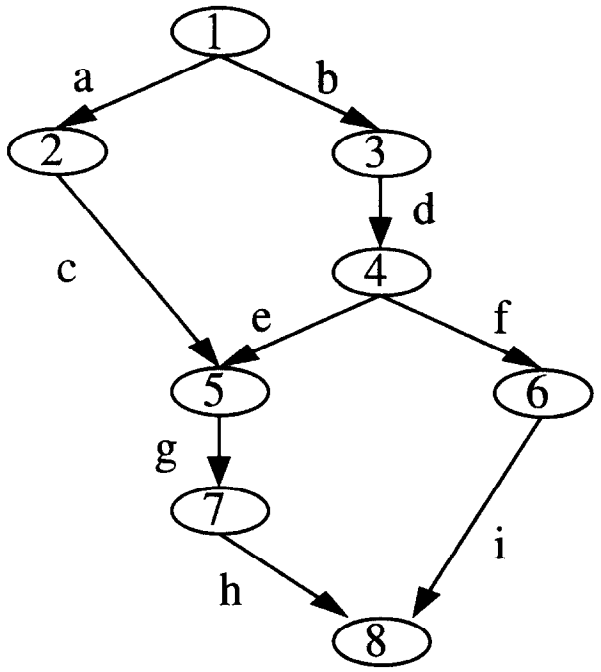

Figure 6. The control flow graph used in experiment one.

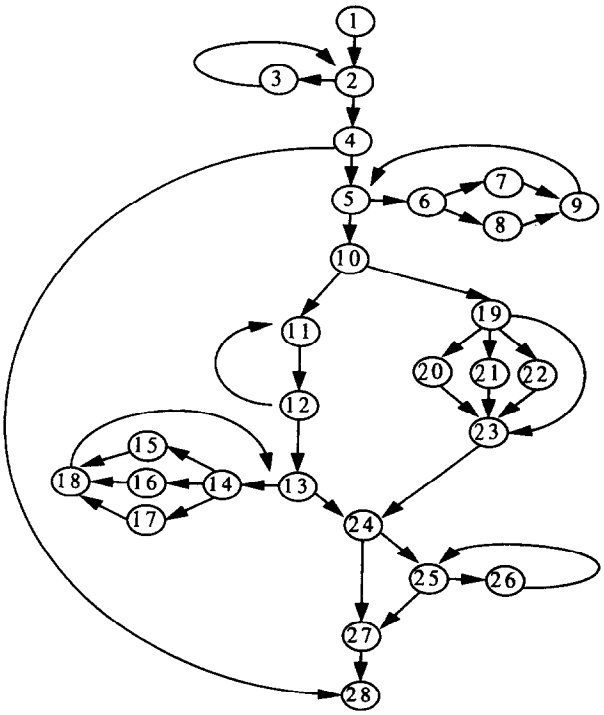

Figure 8. The control flow graph used in experiment four.

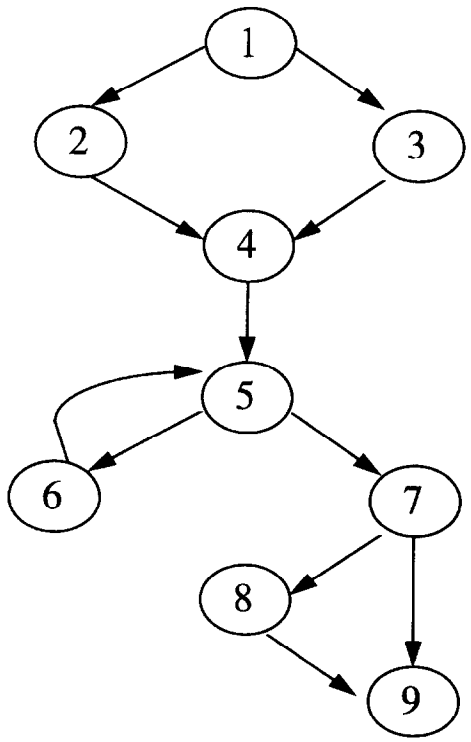

Figure 7. The control flow graph used in experiment three.

lected path sets are reused, the five reduction rules are applied, and then the zero-one optimal path set selection method is applied to efficiently find an optimal path set. (3) If the obtained path set includes infeasible paths, the enhanced approach to handling infeasible paths is applied to speed up the recomputation process. In regression testing, the five reduction rules are applied ahead and then the zero-one optimal path set selection method is used to efficiently find a minimal number of previously executed test cases to fully retest every affected program element at least once.

ACKNOWLEDGMENT

This research was supported by the National Science Council, the Republic of China, under Grant NSC83-0408-E009-047.

REFERENCES

B. Beizer, *Software Testing Techniques*, Data System Analysts Inc., Pennsanken, New Jersey, 1984.

T. Chusho, Test data selection and quality estimation based on the concept of essential branches for path testing. *IEEE Trans. Software Eng.*, vol. SE-13, 509–517, May (1987).

K. F. Fischer, A Test Case Selection Method for The Validation of Software Maintenance Modifications, *IEEE COMPSAC 77* Int. Conf. Procs., Nov. 1977, pp. 421–426.

K. F. Fischer, F. Raji, and A. Chruscicki, A Methodology for Re-testing Modified Software, *Proc. Nat'l Telecomm. Conf.*, CS Press, Los Alamitos, Calif., 1981, pp. B6.3.1-6.

H. N. Gabow, S. N. Maheshwari, and L. J. Osterweil, On Two Problems in the Generation of Program Test Paths. *IEEE Trans. Software Eng.*, vol. SE-2(3), 227–231, Sept. (1976).

Hartmann and D. J. Robson, Techniques for Selective Revalidation. *IEEE Software* 1, 31–36 (1990).

D. Hedley and M. A. Hennell, The Causes and Effects of Infeasible Path in Computer Programs, *IEEE Proceedings of 8th International Conference on Software Engineering*, Aug., 1985, pp. 259–266.

S. Hillier and G. J. Lieberman, *Introduction to Operations Research*, 3rd edition, Holden-Day Inc., California, 1980, pp. 714–740.

S. Y. Hsu and C. G. Chung, A Heuristic Approach to Path Selection Problem in Concurrent Program Testing, in *Proc. the 3rd Workshop on Future Trends of Distributed Computing Systems*, Taipei, Taiwan, April, pp. 86–92, 1992.

W. Krause, R. W. Smith, and M. A. Goodwin, Optimal Software Test Planning Through Automated Network Analysis, *IEEE Proceedings of the 1973 Symposium on Computer Software Reliability*, New York, 1973, pp. 18–22.

J. C. Lin and C. G. Chung, Zero-One Integer Programming Model in Path Selection Problem of Structural Testing, in *IEEE Proceeding COMPSAC 89*, Nov. 1989, pp. 618–627.

M. Marre and A. Bertolino, Unconstrained Duas and Their Use in Achieving All-Uses Coverage, *Proceedings of the 1996 International Symposium on Software Testing and Analysis (ISSTA)*, pp. 147–157, Jan. 1996.

J. McCabe, A Complexity Measure. *IEEE Trans. Software Eng.* vol. SE-2(4), 308–320, Dec. (1976).

E. F. Miller, M. R. Paige, J. B. Benson, and W. R. Wisehart, Structural Techniques of Program Validation, *Digest of Papers, COMPCON*, Feb. 1974, pp. 161–164.

C. Ntafos and S. L. Hakimi, On Path Cover Problems in Digraphs and Applications to Program Testing. *IEEE Trans. Software Eng.* vol. SE-5(5), 520–529, Sep. (1979).

R. E. Prather and J. P. Myers, Jr., The Path Prefix Software Testing Strategy. *IEEE Trans. Software Eng.*, vol. SE-13(7), 761–766, July (1987).

Rapps and E. J. Weyuker, Selecting Software Test Data Using Data Flow Information. *IEEE Trans. Software Eng.*, vol. SE-11(4), 367–375, April (1985).

Schrage, *User's Manual for Linear, Integer, and Quadratic Programming with Lindo*, 3rd ed., The Scientific Press, Palo Alto, CA, 1987.

M. Syslo, N. Deo, and J. S. Kowalik, *Zero-One Integer Programming, in Discrete Optimization Algorithms*, Prentice-Hall, U.S.A., 1983, pp. 100–113.

H. S. Wang, S. R. Hsu, and J. C. Lin, A Generalized Optimal Path-Selection Model for Structural Testing, *Journal of Systems and Software*, 55–63, July (1989).

W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, Effect of test set minimization on fault detection effectiveness. In *Proceedings of the 17th International Conference on Software Engineering (ICSE 95)*, pp. 41–50, April 1995.

APPENDIX: ALGORITHM FOR REDUCTION RULES 2–5

Input: F, m, n, X
F is the coverage frequency matrix obtained after applying reduction rule 1.
m is the number of rows of F.
n is the number of columns of F.
$X(m \times 1)$ is the path number list.
// consider the case that the paths may not be arranged as $p_1, p_2, \ldots, p_m$.

Output: F', m', n', X'
F' is the reduced coverage frequency matrix.
m' is the number of rows of F'.
n' is the number of columns of F'.
$X'(m' \times 1)$ is the reduced path number list.

```
Procedure Rule 2-5(array F, int m, int n, array X, var array F', var int m', var int n', var array X')
{
 int Col_Sum[n] = {0, 0, ..., 0}; // Summation of each column
 int Row_Sum[m] = {0, 0, ..., 0}; // Summation of each row
 For (int row = 1; row ≤ m; row++)
  For (int col = 1; col ≤ n; col++)
  If (f(row,col) ≠ 0)
  {
   Row_Sum[row] = Row_Sum[row] + f(row,col);
   Col_Sum[col] = Col_Sum[col] + f(row,col);
```

```
 }
 // If a row, say i, is deleted, the corresponding Row_Sum[i] will be set zero and each nonzero entry
 // of the ith row will be set zero, too. The same approach is applied to column deletion, too.
 // Since the rows with Row_Sum = = 0 will be skipped in the afterward checking, rule 5 can be
 // handled automatically. The same approach is applied to the columns with Col_Sum = = 0.
 Flag = 1; // Once a reduction rule is activated, set Flag = 1.
 While (Flag = = 1) // Flag = = 1 means at least one reduction rule is activated in the last iteration
{
 Flag = 0;
 (reduction rule 2: essential path)
 For (col = 1; col ≤ n; col + + ) // check each column to find essential path
 If (Col_Sum[col] ≠ 0)
 {
 essential_row = −1;
 For (row = 1; row ≤ m; row + + )
  If (f(row,col) = = Col_Sum[col])
  {
   essential_row = row;
   Break;
  }
 If (essential_row ≠ −1)
 {// delete the columns covered by the essential row
  For (tmp_col = 1; tmp_col ≤ n; tmp_col + + )
   If (f(essential_row,tmp_col) ≠ 0)
    For (row = 1; row ≤ m; row + + )
     If (f(row,tmp_col) ≠ 0)
     {
      Row_Sum[row] = Row_Sum[row] − f(row,tmp_col);
      Col_Sum[tmp_col] = Col_Sum[tmp_col] − f(row,tmp_col);
      f(row,tmp_col) = 0;
     }
  // print out the selected essential path
  Cout ≪ "The" ≪ X[essential_row] ≪ "th path is selected." ≪ "\n";
  Flag = 1;
 }
 }
 (reduction rule 3: dominating column)
 For (col_one = 1; col_one < n; col_one + + )
  For (col_two = col_one + 1; col_two ≤ n; col_two + + )
   If ((Col_Sum[col_two] ≠ 0) & &(Col_Sum[col_one] ≠ 0))
    {
     dominate_flag = 0;
     // The dominate_flag is used to record the dominating relationship
     // between col_one and col_two that have been checked so far.
     // If dominate_flag = = 0, col_one is equal to col_two.
     // If dominate flag = = 1, col_one dominates col_two.
     // If dominate_flag = = 2, col_two dominates col_one.
     // If the dominating relationship does not exist, set dominate_flag = −1 and skip the loop
        For (row = 1; row ≤ m; row + + )
        If (Row_Sum[row] ≠ 0)
        {
         If (f(row,col_one) < f(row,col_two))
         {
          If (dominate_flag = = 0)
           dominate_flag = 1;
          If (dominate_flag = = 2)
          {
           dominate_flag = −1;
```

```
        Break;
        }
        }
        If (f(row,col_one) > f(row,col_two)
        {
       If (dominate_flag = = 0)
        dominate_flag = 2,
       If (dominate_flag = = 1)
       {
        dominate_flag = −1;
        Break;
       }
        }
      }
   If ((dominate_flag = = 1)||(dominate_flag = = 0))
   {// delete col_two
    For (row = 1; row ≤ m; row++)
     If (f(row,col_two) ≠ 0)
     {
      Row_Sum[row] = Row_Sum[row] − f(row,col_two);
      Col_Sum[col_two] = Col_Sum[col_two] − f(row,col_two);
      f(row,col_two) = 0;
     }
    Flag = 1;
   }
   If (dominate_flag = = 2)
   {// delete col_one
    For (row = 1; row ≤ m; row++)
     If (f(row,col_one) ≠ 0)
     {
      Row_Sum[row] = Row_Sum[row] − f(row,col_one);
      Col_Sum[col_one] = Col_Sum[col_one] − f(row,col_one);
      f(row,col_one) = 0;
     }
    Flag = 1;
   }
   }
(reduction rule 4: dominating row)
For (row_one = 1; row_one < m; row_one++)
 For (row_two = row_one + 1; row_two ≤ m; row_two++)
  If ((Row_Sum[row_two] ≠ 0) & &(Row_Sum[row_one] ≠ 0))
   {
   dominate_flag = 0;
   // The dominate_flag is used to record the dominating relationship
   // between row_one and row_two that have been checked so far.
   // If dominate_flag = = 0, row_one is equal to row_two.
   // If dominate_flag = = 1, row_one dominates row_two.
   // If dominate_flag = = 2, row_two dominates row_one.
   // If the dominating relationship does not exist, set dominate_flag = −1 and skip the loop
    For (col = 1; col ≤ n; col++)
    If (Col_Sum[col] ≠ 0)
    {
     If (f(row_one,col) > f(row_two,col)
     {
      If (dominate_flag = = 0)
       dominate_flag = 1;
      If (dominate_flag = = 2)
      {
       dominate_flag = −1;
```

```
      Break;
     }
    }
    If (f(row_one,col) < f(row_two,col))
    {
     If (dominate_flag = = 0)
      dominate_flag = 2;
     If (dominate_flag = = 1)
     {
      dominate_flag = −1;
      Break;
     }
    }
   }
   If ((dominate_flag = = 1)||(dominate_flag = 0))
   { //delete row_two
    For (col = 1; col ≤ n; col + + )
     If (f(row_two,col) ≠ 0)
     {
      Row_Sum[row_two] = Row_Sum[row_two] − f(row_two,col);
      Col_Sum[col] = Col_Sum[col] − f(row_two,col);
      f(row_two,col) = 0;
     }
    Flag = 1;
    }
    If (dominate_flag = = 2)
    { // delete row_one
    For (col = 1; col ≤ n; col + + )
     If (f(row_one,col) ≠ 0)
     {
      Row_Sum[row_one] = Row_Sum[row_one] − f(row_one,col);
      Col_Sum[col] = Col_Sum[col] − f(row_one,col);
      f(row_one,col) = 0;
     }
    Flag = 1;
    }
   }
}// End of While(Flag = = 1)
// Copy undelete (i.e., non-empty) rows and columns of F to F',
// Copy undeleted rows of X to X'
m' = 0;
For (int row = 1; row ≤ m; row + + )
 If (Row_Sum[row] ≠ 0)
  m' = m' + 1;
n' = 0;
For (int col = 1; col ≤ n; col + + )
 If (Col_Sum[col] ≠ 0)
  n' = n' + 1;
F' = new array[m',n'];
X' = new array[m'];
int new_row = 1;
For (old_row = 1; old_row ≤ m; old_row + + )
 If (Row_Sum[old_row] ≠ 0)
 {
  X'[new_row] = X[old_row];
  int new_col = 1;
  For (old_col = 1; old_col ≤ n; old_col + + )
   If (Col_Sum[old_col] ≠ 0)
   {
```

```
    f'(new_row,new_col) = f(old_row,old_col);
    new_col = new_col + 1;
   }
  new_row = new_row + 1;
 }
}// End of the procedure
```