OPTIMIZATION PROBLEMS FROM FEATURE TESTING OF COMMUNICATION PROTOCOLS

David Lee (lee@research.att.com)
Mihalis Yannakakis (mihalis@research.att.com)

Bell Laboratories
Murray Hill, New Jersey

ABSTRACT

In feature testing of communication protocols, we want to construct a minimal number of tests with a desirable fault coverage. We model the protocols by extended finite state machines and reduce the test generation process to optimization problems on graphs. We study efficient algorithms and their complexity. We report experimental results on real systems, including Personal HandyPhone System, a 5ESS based ISDN wireless system, and 5ESS Intelligent Network Application Protocol.

1. INTRODUCTION

A complex communication system uses a set of rules, called a *protocol*, to define the interactions among the system entities. To ensure that they communicate reliably a protocol implementation must be tested for conformance to its specification. Typically, a system is tested by an external tester; the tester applies a selected input sequence to the system and verifies that the corresponding output sequence is that which is expected. It is impossible to test all possible input sequences; we want to perform a minimal number of tests with a desirable fault coverage.

As a protocol is specified and implemented, testing is conducted at different stages. There are lower level tests with a variety of names such as deliverable or developer's tests. There are high level tests such as *feature* testing. We shall focus on feature testing, study several theoretical problems, develop efficient algorithms, and report the experimental results on real systems.

A protocol specification is typically broken into control and data portions where the control portion is modeled by an ordinary finite state machine (FSM) [Ko, LY2]. Most of the formal work on conformance testing addresses the problem of testing the control portion [ADLU, BU, CZ, DSU, FBKAG, Go, KSNM, LY2, MP2, MP3, SD, SL, YL1]. Machines that arise in this way usually have a relatively small number of states (from one to a few dozen), but a large number of different inputs and outputs (50 to 100 or more). For example, the IEEE 802.2 Logical Link Control Protocol (LLC) [ANSI] has 14 states, 48 inputs (even without counting parameter values) and 65 outputs. Clearly, there is an enormous number of machines with that many states, inputs, and outputs, and, consequently, brute force testing is infeasible. A number of methods have been proposed which work for special cases (such as, when there is a distinguishing sequence or a reliable reset capability), or are generally applicable but may not provide a complete fault coverage. For instance, there are the D-method based on distinguishing sequences [He], the U-method based on UIO sequences [ADLU, SD], the W-method based on characterization sets [Ch], and the T-method based on transition tours [NT]. A general polynomial time test generation procedure was reported recently in [YL1]. For a survey, see [Ko, LY2, SL].

Finite state machines model well control portions of communication protocols. However, the data portions include variables and operations based on their values; ordinary finite state machines are not powerful enough to model in a succinct way the physical systems any more. Extended finite state machines, which are finite state machines extended with variables, have emerged from the design and analysis of both communication protocols [Ho, LY2] and sequential circuit study [CK]. For instance, IEEE 802.2 LLC is specified by 14 control states, a number of variables, and a set of transitions (pp. 75-117). For example, a typical transition is (p. 96):

current_state SETUP
input ACK_TIMER_EXPIRED
predicate S_FLAG = 1
output CONNECT_CONFIRM
action P_FLAG := 0; REMOTE_BUSY := 0
next_state NORMAL

In state SETUP and upon input ACK_TIMER_EXPIRED, if variable S_FLAG has value 1, then the machine outputs CONNECT_CONFIRM, sets variables P_FLAG and REMOTE_BUSY to 0, and moves to state NORMAL.

To model this and other protocols, including other ISO standards and complicated systems such as 5ESS[1], we extend finite state machines with variables as follows. We denote a finite set of variables by a vector: $\vec{x} = (x_1, \cdots, x_k)$. A predicate on variable values $P(\vec{x})$ returns FALSE or TRUE; a set of variable values $\vec{x}$ is valid for P if $P(\vec{x})$ = TRUE, and we denote the set of valid variable values by $X_P = \{\vec{x} : P(\vec{x}) = \text{TRUE}\}$. Given a function $A(\vec{x})$, an action (transformation) is an assignment: $\vec{x} := A(\vec{x})$.

Definition 1. An *extended finite state machine* (EFSM) is a quintuple

$$M = (I, O, S, \vec{x}, T)$$

where I, O, S, $\vec{x}$, and T are finite sets of input symbols, output

[1] AT&T No. 5 Electronic Switching System

0-8186-7453-9/96 $05.00 © 1996 IEEE

symbols, states, variables, and transitions, respectively. Each transition t in the set T is a 6-tuple:

$$t = (s_t, q_t, a_t, o_t, P_t, A_t)$$

where s_t, q_t, a_t, and o_t are the start (current) state, end (next) state, input, and output, respectively. $P_t(\vec{x})$ is a predicate on the current variable values and $A_t(\vec{x})$ defines an action on variable values.

Initially, the machine is in an initial state $s_0 \in S$ with initial variable values: $\vec{x}_{init}$. Suppose that the machine is at state s_t with the current variable values $\vec{x}$. Upon input a_t, if $\vec{x}$ is valid for P_t, i.e., $P_t(\vec{x})$ = TRUE, then the machine follows the transition t, outputs o_t, changes the current variable values by action $\vec{x} := A_t(\vec{x})$, and moves to state q_t.

For each state $s \in S$ and input $a \in I$, let all the transitions with start state s and input a be: $t_i = (s, q_i, a, o_i, P_i, A_i)$, $1 \le i \le r$. In a *deterministic* EFSM the sets of valid variable values of these r predicates are mutually disjoint, i.e., $X_{P_i} \cap X_{P_j} = \emptyset$, $1 \le i \ne j \le r$. Otherwise, the machine is *nondeterministic*.

□

In a deterministic EFSM there is at most one transition to follow at any moment, since at any state and upon each input, the associated transitions have disjoint valid variable values for their predicates and, consequently, current variable values are valid for at most one predicate. On the other hand, in a nondeterministic EFSM there may be more than one possible transition to follow. In this paper we only consider deterministic EFSM's. Clearly, if the variable set is empty and all predicates $P \equiv$ TRUE then an EFSM becomes an ordinary FSM.

Each combination of a state and variable values is called a *configuration* (or *context*). Given an EFSM, if each variable has a finite number of values (Boolean variables for instance), then there is a finite number of configurations, and hence there is an equivalent (ordinary) FSM with configurations as states. Therefore, an EFSM with finite variable domains is a compact representation of an FSM.

Our work was motivated by the efforts in test generation for Personal HandyPhone System (PHS), which is a 5ESS based ISDN wireless system, a product of AT&T Network Systems, and also for 5ESS Intelligent Network Application Protocol (INAP). It turns out that both systems can be properly modeled by EFSM's and we want to construct a minimal number of test sequences with a complete fault coverage. We first discuss general theory and efficient algorithms and then report experiment results on the two systems.

An EFSM usually has an initial state s_0 and all the variables have an initial value $\vec{x}_{init}$, which consists of the *initial configuration*. A *test sequence* (or a *scenario*) is an input sequence that takes the machine from the initial configuration back to the initial state (possibly with different variable values). We want to construct a set of test sequences of a desirable *fault coverage*, which ensures that the implementation machine under test *conforms* to the specification. The fault coverage is essential. However, it is often defined differently from different models and/or practical needs. For testing FSM's we may need a *checking sequence* [Ch, He, Ko, LY2, Mo, Va, YL1], which guarantees that the implementation machine is structurally isomorphic to the specification machine. However, even for medium size machines it is too long to be practical [YL1] while for EFSM's hundreds of thousands of states (configurations) are typical and it is in general impossible to construct a checking sequence.

From the experiences of our testing experts for the two systems, PHS and INAP, a practical criterion of fault coverage is that each transition in the specification EFSM has to be executed at least once: [2]

Definition 2. A *complete test set* for an EFSM is a set of test sequences such that each transition is tested at least once.

□

Given the succinct representation of EFSM's, one might imagine that it is an *easy* problem. As a matter of fact, even an apparently easier problem, the reachability problem, is hard where we want to determine if a control state is reachable from the initial state. Specifically, the Turing machine halting problem can be reduced to the reachability problem of EFSM, which is, therefore, undecidable if the variable domains are infinite and PSPACE-complete otherwise [HU]. Recently, testing of EFSM's has becoming an important topic, especially in the network protocol area [ML, MP1, KL].

To find a complete test set, we first construct a *reachability graph G*, which consists of all the configurations and transitions that are reachable from the initial configuration. We obtain a directed graph where the nodes and edges are the reachable configurations and transitions, respectively. Obviously, a control state may have multiple appearances in the nodes (along with different variable values) and each transition may appear many times as edges in the reachability graph. In this reachability graph, any path from the initial node (configuration) corresponds to a feasible path (test sequence) in the EFSM, since there are no predicate or action restrictions anymore. Therefore, a set of such paths in G, which exercises each transition at least once, provides a complete test set for the EFSM. We thus reduce the testing problem to a graph path covering problem.

The construction of the reachability graph is often a formidable task; it has the well-known state explosion problem due to the large number of possible combinations of the control states and variable values. One approach to this problem is to apply an on-line minimization algorithm to construct an equivalent graph $G_{\min}$, which collapses all configurations of the reachability graph that are equivalent in terms of the transitions that they can perform. Such a minimized graph can be constructed efficiently directly from the EFSM [LY1]; $G_{\min}$ could be much smaller than G and can be used in its place for generating test sequences. Furthermore, for the testing purpose, we do not need a complete reachability graph; we only need a subgraph that contains all the transitions so that a set of covering paths still provides a complete test set [HLS]. We shall not digress to this

[2] For INAP testing, the criterion is different: each piece of the specification code has to be executed at least once. Our algorithms are applicable to both systems.

topic further. From now on we assume that we have a graph G that contains all the transitions of a given EFSM and we want to construct a complete test set of a small size. For clarity, we assume that each path (test sequence) is from the initial node to a *sink* node, which is a configuration with the initial control state.

To summarize, we have a directed graph with an initial node and a sink node. The nodes are configurations, which correspond to combinations of control states and variable values, and a state may appear in more than one node. The edges correspond to transitions of the EFSM, and the same transition may appear many times in the graph as edges between different configurations. We want to find a complete test set: a set of paths from the initial node to the sink node such that each transition in the *original* EFSM is covered; specifically, among the multiple appearances of a transition, it is sufficient to cover any one of them. Therefore, the test generation is reduced to covering path problems on graphs.

2. PATH CONSTRUCTION

Formally, we have a directed graph $G = <V, E>$ with $n = |V|$ nodes, $m = |E|$ edges, a *source* node s of in-degree 0, and a *sink* node t of out-degree 0. All edges are reachable from the source node and the sink node is reachable from all edges. There is a set C of $k = |C|$ distinct *colors*. Each node and edge is associated with a subset of colors from C [3]. A path from the source to sink is called a *test*.

We are interested in a set of tests that cover all the colors; they are not necessarily the conventional covering paths that cover all the edges. Formally, a *complete test set* covers all the colors in C. The path (test) length makes little difference and we are interested in minimizing the number of paths. We shrink each strongly connected component into a node, which contains all the colors of the nodes and edges in the component. The problem then is reduced to that on a directed acyclic graph (DAG). From now on, unless otherwise stated, we assume that the graph $G = <V, E>$ is a DAG.

2.1. Minimal Complete Test Set

We need a complete test set - a set of paths from the initial node to the sink node that cover all the colors. On the other hand, in the feature testing of communication systems, setting up and running each test is time consuming and each test is costly to experiment. Consequently, we want to minimize the number of tests:

Problem 1: Find a complete test set of minimum cardinality.

The problem is NP-hard; the *Set Cover* problem [Ka, GJ, p. 222], which is known to be NP-hard, can be reduced to this problem. Recall the Set Cover problem: given a family F of sets over a (finite) set U of elements and an integer κ, is there a covering subfamily of no more than κ members, i.e. are there κ sets in F whose union is U? In the reduction we let U be the set of colors, We construct a graph with two nodes: the source and sink, and parallel edges from source to sink such that each set in F is associated with an edge. A solution of Problem 1 provides an answer to the Set Cover problem. This construction uses parallel edges and has multiple colors associated with an edge, but it is easy to modify it, if we wish, so that the graph is simple (i.e. has no parallel edges) and each edge has only one associated color: simply replace the edges by paths of appropriate length.

We can solve Problem 1 in time $m2^{O(k)}$, i.e. exponential in the number of colors and linear in the size of the graph, by a bottom-up, essentially brute-force approach. Since G is a DAG, we can topologically sort the nodes (see eg. [CLR]). Compute a family F_u of color sets for each node u in the reverse topological order as follows. The sink node contains a singleton set of all its colors. When processing a node u, we examine all its outgoing edges (u, v); note that node v has been processed and has already a family F_v of color sets. For each set of F_v, we add the colors of edge (u, v) and that of node u, and obtain a set of colors. We collect all the color sets from all the outgoing edges from u and obtain a family of color sets for u. We can obviously remove duplicates, so $|F_u| \le 2^k$.

Besides duplicates, we can also remove a color set which is included in another one, i.e., if there are two colors sets C_i and C_j at a node u with $C_i \subseteq C_j$ then we can discard C_j. Therefore, at each node, we only have to a keep a record of color sets, none of which is a subset of the other. Given a set C of cardinality $k = |C|$, denote by $\mu(k)$ the maximum number of subsets of C such that none is a subset of another. It is well known (see eg. [Lo2]) that the maximum is achieved by the family of all subsets of cardinality $\lfloor k/2 \rfloor$ and $\mu(k) = \binom{k}{\lfloor k/2 \rfloor} = \Theta(\frac{2^k}{\sqrt{k}})$.

Obviously, the family F_u of color sets of u has the following properties: (1) For each color set, there is a path from u to the sink whose colors are the same as the color set; and (2) The colors on any path from u to the sink are contained in one of the color sets in u. This can be easily proved by an induction on the ordering of processing. For an efficient path construction from the color sets, we associate edge (u, v) to the color sets of u, which are the unions of the initial color set of u, the initial color set of (u, v), and the computed color sets of v, so that we can trace the path corresponding to each color set of u.

At the source node s, we find a minimal number of color sets in F_s that covers all the colors. The corresponding paths can be constructed from the edges associated with the color sets. This gives a complete test set of minimum cardinality.

We discuss the time complexity. When we process a node u, for each outgoing edge (u, v), we merge the color sets associated with node u, and edge (u, v), with every set of the family F_v for node v, which has been processed. Although the color sets that are contained in other sets are not needed and can be discarded, it takes some effort to find and eliminate them. We can choose to apply the algorithm either with the elimination of subsets at the nodes, or without subset elimination (i.e.

[3] Each transition in the EFSM corresponds to a distinct color in C and may have multiple appearances in G. We consider a more general case here; each node and edge have a set of colors from C.

only duplicate elimination). Let f, f' respectively be an upper bound on the number of color sets at each node in these two cases; thus, $f \le f'$, $f \le \mu(k)$ and $f' \le 2^k$, where k is the number of colors. We can represent a color set by a list of its elements or by its characteristic vector. We can represent a family of color sets (eg. F_u) by a trie. Checking whether a color set is in a family F and adding it if it is not takes time $O(k)$. Thus, if we do not eliminate subsets, then the operation on each edge takes time proportional to $f'k$ and the total cost is $O(f'km)$, where m is the number of edges of the graph. If we eliminate subsets, then the straightforward implementation of each edge operation takes time proportional to f^2k and the total cost is $O(f^2km)$. The quadratic factor is because for each set that is added to a family we have to compare it for inclusion with every set of the family. Unfortunately there is no good way known to find the maximal sets of a given family (i.e., those that are not contained in another set) in time substantially better than the straightforward quadratic time; a small improvement (by a factor of $\sqrt{\log f}$) is given in [YJ]. Subset elimination is advantageous only if $f^2 < f'$. Note that for the worst case upper bounds of f and f' we have $\mu^2(k) >> 2^k$. In any case, without subset elimination we can compute the family F_s for the root s in worst case time complexity $O(2^k km)$ over the whole graph.

At the source node s, we select a minimal number of color sets from the family F_s that cover all the k colors. This can be done in time $O(2^k f'k) = 2^{O(k)}$. Consider a graph that has one node for every subset of colors, and has an edge from node P to node Q if $Q = P \cup R$ for some $R \in F_s$. The graph has 2^k nodes and out-degree f', thus it has $2^k f'$ edges. A minimum cover corresponds to a shortest path in the graph from node $\emptyset$ to the node C of all the colors. The standard breadth-first search algorithm will find the shortest path in linear time in the size of the graph. Once the minimum cover is found, the corresponding paths of the DAG can be traced from the edges associated with the color sets with a cost proportional to the total lengths of the paths.

2.2. Maximal Color Paths

The cost of the above approach is exponential in the number of colors k and thus it is feasible only for small values of k. For moderate and large number of colors we need to restrict ourselves to approximation algorithms. Our problem is a Set Cover problem where the sets are not given explicitly, but rather are defined implicitly by the paths of the graph. The standard approximation algorithm for Set Cover [Jo, Lo1] is the following greedy method: pick a set of maximum cardinality from the given family, then pick a set that covers the maximum number of uncovered elements, and continue similarly picking sets until all the elements are covered. This algorithm corresponds to the following greedy method for our problem. We first find a path (test) that covers a maximum number of colors and delete the covered colors from C. We then repeat the same process until all the colors have been covered. Thus, we have the following problem:

Problem 2: Find a test that covers the maximum number of colors.

This problem is also NP-hard. Our reduction is from the maximum independent set problem in regular graphs. In this problem we are given a regular undirected graph H (i.e. all the nodes have the same degree d) and a positive integer l, and we wish to determine if there is a independent set I of at least l nodes (i.e., no two nodes of I are adjacent). Construct a DAG G that consists of a source node s, sink node t, and l levels of n nodes each, where n is the number of nodes of H. Thus, every node u of G (except s and t) in each level corresponds to a node $h(u)$ of H. There are edges from s to all the nodes of the first level, edges from all the nodes of each level to all the nodes of the next level, and finally edges from the nodes of the last level l to node t. We let the set of colors be the set of edges of the given graph H. The set of colors associated with all edges of G and with nodes s and t is $\emptyset$, and the set of colors associated with any other node u of G is the set of edges of H incident to the corresponding node $h(u)$ of H.

We claim that H has an independent set with l nodes if and only if G has a path that covers dl colors. From the construction it is clear that there is a 1-to-1 correspondence between $s-t$ paths of G and sequences of l (not necessarily distinct) nodes of H. The set of colors covered by a path in G is the set of edges of H incident to the nodes of the corresponding sequence. Since H is a regular graph of degree d, such a set of edges (colors) can have at most cardinality dl, and furthermore, it has cardinality exactly dl iff the l nodes of H do not share any edge (implying in particular that they are distinct), i.e., iff the l nodes form an independent set.

Thus, in view of the NP-hardness of Problem 2 we have to content ourselves with approximation algorithms. We now describe some simple heuristic methods.

1. Longest Path

Suppose that an edge (node) has c uncovered colors so far. We assign a weight c to that edge (node), and we have a weighted graph. Find a longest path from the source to sink; it is possible since the graph is a DAG. This may not provide a maximal color test due to the multiple appearances of colors on a path. However, if there are no multiple appearances of colors on the path, then it is indeed a maximal color test.

There are known efficient ways of finding a longest path on a DAG. We can first topologically sort the nodes and then compute the longest paths from each node to the sink in the reverse topological order. Specifically, suppose that we are processing node u and examine all its outgoing edges (u, v) where v is a node of higher topological ordering and has its longest path to the sink computed. Suppose that (u, v) has weight $w_{u,v}$ and that a longest path from v to sink has weight w_v. Then a path from u to v and then following a longest path from v to the sink has a weight $w_{u,v} + w_v$. We can easily compare all the outgoing edges from u and find a longest path from u to the sink node. The time and space needed is $O(m)$ where m is the number of edges.

How does this heuristic method compare with the optimal solution? In the worst case it can be really bad.

Example 1

A graph G consists of $k+1$ separate paths P_i, $i = 0, 1, \cdots, k$, from source to sink. Path P_0 has k edges of k distinct colors and path P_i has $k+1$ edges, all of the same color i, $i = 1, \cdots, k$. Path P_0 provides a minimal complete test set with only 1 test. However, the longest path heuristic will pick one of the paths with $k+1$ edges. Iterating the process, the method will require k tests corresponding to the k paths $P_i, i = 1, \cdots, k$.

□

2. A Greedy Heuristic

In Section 2.1, we have described an exact procedure for finding a maximal color path from source to sink in time $O(2^k km)$. We now discuss a greedy heuristic procedure based on a similar idea, yet it takes linear time and works well in practice, see Section 5.

We again topologically sort the nodes and compute a desired path from each node to the sink in a reverse topological order as follows. Instead of keeping the color sets of all the paths from a node to the sink, we only keep the one with a supposedly "maximum number" of colors. Specifically, when we process a node u and consider all the outgoing edges (u, v) where v has a higher topological order and has been processed, we take the union of the colors of node u, edge (u, v), and node v. We compare the resulting color sets from all the outgoing edges from u and keep one with the largest cardinality. This procedure is well defined since G is a DAG. However, it may not provide a maximum color coverage test; when we choose the outgoing edge from u, we do not incorporate information of the colors from the source to u. Since we take unions of and compare color sets of no more than k colors, the time and space complexity of this approach is $O(km)$.

Although the second method seems to be better in many cases, its worst case ratio to the optimal solution is also $\Omega(k)$.

Example 2

Consider the DAG G shown in Figure 1. The colors 1,..., 6 associated with the edges are shown next to them, and the nodes have no colors. The optimal path $v_7, \ldots, v_1$ covers all the colors.

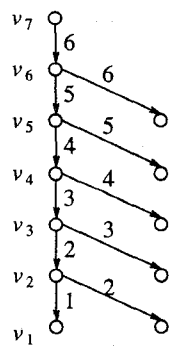

Figure 1

Consider the application of the greedy heuristic to this graph. When we process node v_2 we have a choice whether to keep color set {1} or {2}. We have not specified how to break ties. Suppose that ties are broken in favor of higher numbered colors. Then the heuristic will pick {2} for v_2. At node v_3, there is a choice between {2} and {3}, so we will pick set {3}. Continuing in this manner, we see that the heuristic will find a set of cardinality 1 instead of 6. The example clearly generalizes to arbitrary number of colors k. □

It is easy to improve on the greedy heuristic. For example if we keep at each node two sets instead of only one, then the heuristic will find the optimal path in Example 2, and in general it will be guaranteed to find a path of at least two colors if there is one (see Section 4). However, it is easy to construct another example for this 2-set greedy variant in which it has ratio $k/2$ to the optimal, i.e., it finds only a path of two colors while the optimal path covers all the colors. Another variant of the greedy algorithm does somewhat better.

3. A Transitive Greedy Heuristic

This is similar to the greedy heuristic, except that when we process a node u, we do not consider only its immediate successors but all its descendants. Specifically, for each outgoing edge (u, v) and descendant v' of v (possibly $v = v'$), we take the union of the colors of node u, edge (u, v), and node v'. We compare the resulting color sets from all the outgoing edges from u and descendants v' and keep one with largest cardinality. The time complexity of this algorithm is $O(knm)$.

The worst case ratio of this heuristic to the optimal is somewhat better — it is $\Theta(\sqrt{c})$, where c is the maximum number of colors covered by a path. To prove the upper bound, consider an optimal $s-t$ path p that covers c colors. For each node u, let $F(u)$ denote the set of colors assigned to u by the algorithm. Traversing the path p bottom-up from the sink t to the source s, let $u_0 = t$, let $u_1, u_2, \ldots, u_r$ be the nodes at which the cardinality of F strictly increases, and if $u_r \neq t$ let $t = u_{r+1}$; i.e. for $j = 1, \ldots, r$, $|F(u_j)| > |F(u_{j-1})|$ and $|F(u)| = |F(u_{j-1})|$ for all nodes u of p between u_{j-1} and u_j. Let C_j be the set of colors that appear on the edges and nodes of p between u_{j-1} and u_j, where node u_{j-1} is excluded and u_j is included. Observe that, if the segment of p between u_{j-1} and u_j is nontrivial, i.e., contains some intermediate nodes and edges, then all their colors must be contained in $F(u_{j-1})$, because otherwise one of these intermediate nodes would be assigned a larger color set. Thus, all colors of $C_j - F(u_{j-1})$ appear on the node u_j and the edge from u_j to its child in p. From the computation of $F(u_j)$ we have, $|F(u_j)| \geq |F(u_{j-1}) \cup (C_j - F(u_{j-1}))| \geq |C_j|$ Note that the number c of colors of the path p is equal to $|F(u_0) \cup \bigcup_j C_j| \leq \sum_j |F(u_j)|$. Let $f = |F(t)|$ be the number of colors obtained by the heuristic. Since the sets $F(u_j)$ have distinct cardinalities, except possibly for the last two, we have $c \leq \sum_{j=1}^{f+1} j = (f+1)(f+2)/2$, and thus $f = \Omega(\sqrt{c})$. It is not hard to construct an example as in Figure 1 showing that the analysis is tight, i.e., an example where $f = O(\sqrt{c})$. □

However, it is worth noting that the greedy heuristic achieves a constant factor (no more than 2) approximation in the DAGs of the NP-hardness reduction; we omit the proof. (The longest path heuristic does not have this property; it does not provide any guarantees.) This is particularly interesting in view

of the fact that even for this class of DAGs there is a limit on the approximation factor that can be achieved in polynomial time. This follows from the fact that the maximum independent set problem on regular graphs of bounded degree d is MAX SNP-hard, and hence there is a constant $r > 1$ such that it is NP-hard to distinguish for a given graph H and integer l between the following two cases: (1) the case where there is an independent set of size at least l and (2) when the maximum independent set has size less than l/r [ALMSS, PY]. Consider the DAG G that is constructed in the reduction from H and l. In case (1), the optimal path covers dl colors. In case (2) it covers less than $dl/r + (d-1)l(r-1)/r$. Thus, if we can approximate the maximum number of colors of a path in G within a factor $1-(r-1)/rd$ then we can distinguish between case (1) and (2) in H.

It follows that there is a $\varepsilon > 0$ (eg, $\varepsilon = (r-1)/rd$) such that Problem 2 cannot be approximated in polynomial time with ratio $1-\varepsilon$ unless P=NP. In other words, Problem 2 does not have a polynomial time approximation scheme. We do not know whether Problem 2 can actually be approximated within some constant factor for all instances.

We now come back to the original minimum complete test set problem (Problem 1). Suppose that we successfully find a maximum color test repeatedly until we obtain a complete test set in N steps and that the minimum complete test set contains N^* tests. How far is N from N^*? Is there a better algorithm? It follows from results on the Set Cover problem that $N = \Theta(N^* \log k)$ [Lo1, Jo]. That is, on the one hand, for any instance, if we can find repeatedly maximum color tests, then the complete test set will contain at most $N^* \log k$ tests; moreover, an approximation within factor r for Problem 2 will yield a test set of size at most $N^* r \log k$. Conversely, there are instances in which even if we could find repeatedly paths that cover the maximum number of colors, the resulting test set contains $N^* \log_e k$ test (where $\log_e$ denotes the natural logarithm).

Moreover, the negative results on the approximation of the Set Cover problem [LuY, Fei] imply that we cannot do better than a logarithmic factor in polynomial time. That is, for any polynomial time algorithm which constructs a complete test set of cardinality N, there are cases such that $N = \Omega(N^* \log k)$ (unless of course P=NP).

3. PATHS WITH A CONSTANT BOUND ON THE NUMBER OF COLORS COVERED

In spite of the negative results in the worst case, the longest path and greedy heuristic procedures were applied to real systems (see Section 5) and proved to be surprisingly efficient; a few tests cover a large number of colors and, afterwards, each test covers a *very small* number of colors. A typical situation is that the first 20% tests cover more than 70% of the colors. Afterwards, 80% of the tests cover the remaining 30% of the colors, and each test covers 1 to 3 colors. Consequently, the costly part of the test generation is the second part. Under these circumstances, exact procedures for either maximal color paths or minimal complete test sets are needed to reduce the number of tests as much as possible. The question is, can we obtain more efficient algorithms if we know that there is a bound on the maximum number of colors on any path that is a small constant $c \ll k$. We consider the following problems.

Problem 3. Suppose that a maximum color test covers no more than $c \ll k$ colors where c is a small constant. (1) Find a minimum complete test set; and (2) Find a maximum color test.

Since a maximum color test covers at most c colors, we can use the exact brute-force method as described in Section 2.1. In this case, each color set at a node contains at most c colors. (If a set contains c colors then we have obtained a solution for Problem 3(2).) Therefore, the number f of color sets recorded at each node is bounded by:

$$\mu(k, c) = \binom{k}{c} = \frac{k!}{c!(k-c)!} \simeq k^c .$$

It takes time $O(k^{2c}m)$ and space $O(k^{2c}n)$ to construct the color sets for all nodes and we can examine the source node and find a maximal color test. Therefore, Problem 3(2) can be solved in time and space polynomial in the number of colors k and the size of the graph. Although the time is polynomial, c appears in the exponent with base k, and so for moderate values of k (say 100) and small values of c (eg. 5), the time is large. There is a better method to find the maximum color set. We will see later that Problem 3(2) can be solved more efficiently, essentially in linear time in the size of the graph for constant c.

First, let us discuss Problem 3(1). Once we have computed the family of color sets at the source node, we need to solve the Set Cover problem to find a subset of minimum cardinality that covers all the k colors. The complexity varies with the constant c.

For $c = 1$, the problem is trivial: since a color set (path) contains at most one color, we can simply take k distinct color sets, which provides a minimum complete test set. On the other hand, at each node we can use a bit map to record the color sets and it takes time $O(k)$ to process each outgoing edge from a node. Therefore, the total time and space complexity is $O(km)$.

For $c = 2$, Problem 3(1) can be solved in polynomial time as follows. Each color set contains 1 or 2 colors. None of the singleton sets is contained in another set; otherwise, they would have been removed during the process. We take all of them into the test sets. Suppose that there are $\bar{k}$ remaining colors to be covered by the two-color sets. We want to select a minimal number of them to cover these $\bar{k}$ colors. We construct an undirected graph as follows. Each of the $\bar{k}$ colors corresponds to a node and each of the two-color sets corresponds to an edge that joins the nodes corresponding to its colors. The problem then is to find a minimal set of edges that cover all the nodes. It is the well-known edge cover problem, which can be solved in polynomial time [GJ, p. 79] by graph matching [La].

For $c \geq 3$, Problem 3(1) is NP-hard; this follows from the NP-hardness of the Set Cover problem even if all the sets have at most 3 elements [GJ].

In the remainder of this section we will describe an efficient algorithm for Problem 3(2), i.e. determining whether the graph contains a path that covers c colors, and finding such a

path. We use techniques similar to those of [Mo] and [AYZ] for finding simple paths in graphs.

If all we want to do is to find a path that covers c colors (rather than all paths), then in the bottom-up computation we do not need to keep all the color sets but only a sufficient number of them. That is, at each node u, instead of the complete family F_u of color sets of the paths starting at u, we need keep only a subfamily L_u such that if the DAG contains a path through u that covers c colors, then there is such a path whose suffix from u to the sink t uses only colors from some member of L_u.

Let us look first at some simple cases of small c.

Case $c=2$. If F_u contains a set with two elements, then we are done; this set suffices. If all sets are singletons, then it suffices to keep in L_u only two of them (in case F_u has more than two). For, suppose that there is a path through u that covers two colors, say a path p_1 from s to u that covers color a and a path p_2 from u to t that covers color b. At least one of the sets in L_u is not $\{a\}$, and thus we can combine the corresponding $u-t$ path with the $s-t$ path p_1 to obtain a path through u that covers 2 colors.

Case $c=3$. If F_u contains a set with 3 colors then we are done. Otherwise, we claim that we need to keep at most only 3 singleton sets and 3 pairs, thus 6 sets in all. If F_u has more than 3 singletons, then just keep any 3 of them. As far as pairs are concerned, we delete a pair $\{a,b\}$, unless there is a color $c\neq a,b$ that is contained in all the remaining pairs, i.e., they are all of the form $\{c,x\}$. Suppose that we cannot delete the pair $\{a,b\}$ and there are 3 or more other pairs besides $\{a,b\}$. Then one of them is $\{c,x\}$ with $x\neq a,b$. In that case, we can delete all the pairs except for $\{a,b\}$ and $\{c,x\}$. Therefore, if no pair can be deleted, then there can be at most three pairs.

To see why the above reduction works, consider a path through u that covers 3 colors, and let p_1 and p_2 be the prefix and suffix before and after u. If p_1 covers two colors a,b, then L_u contains a singleton other than a and b. Suppose p_1 covers one color c, and p_2 colors a and b. If the pair $\{a,b\}$ was deleted, then still a pair remains that does not contain c, and hence its corresponding $u-t$ path can be combined with path p_1.

Case: general c. For general values of c, we need to keep at most c singleton sets, $\binom{c}{2}$ pairs, $\binom{c}{3}$ triples, ..., c sets of size $c-1$, thus a total of at most 2^c-2 sets. To see this consider the sets of F_u of size r, $r=1,\ldots,c-1$. We can delete such a set unless there is a disjoint set of $c-r$ colors that intersects all the remaining sets of F_u of size r. Thus, if we cannot delete any more sets of size r, then we have a family of r-sets $S_1,\ldots,S_q$ and there are corresponding $(c-r)$-sets $T_1,\ldots,T_q$, such that $E_i \cap T_i = \emptyset$ for all $i=1,\ldots,q$, and $E_i \cap T_j \neq \emptyset$ for all $i\neq j$. The number q of such sets can be at most $\binom{c}{r}$; see eg. [Lo2, Ex. 13.32].

To apply the above reduction process we need to determine at each node which sets are unnecessary and delete them. For small values of c we can do this easily (as we saw for $c=2$ and $c=3$), and thus obtain an algorithm that runs in linear time in the size of the input (the DAG and the given color sets for the nodes and edges). For general values of c however this is not so easy. Unfortunately, the proof of the fact in [Lo2] is nonconstructive.

We describe now an algorithm that gets around this problem. We use the color coding method of [AYZ]. Suppose the given colored DAG contains a path p that covers c colors. Consider a random mapping h from the set of k colors to the set $\{1,\ldots c\}$. For any set S of c distinct colors (in particular the set of colors covered by p), the restriction of h to S is one-to-one with probability $c!/c^c=\Theta(\sqrt{c}e^{-c})$. Applying a mapping h yields a new recoloring of the DAG with c colors. We can use the straightforward algorithm on the c-colored DAG to search in time $O(2^c m)$ for a path that covers all c colors; such a path will be found if the mapping h is one-to-one for some set S of c original colors that is covered by a path in the DAG, i.e., with probability at least $c!/c^c$. Therefore, if we apply the above procedure for a sequence of random mappings h, we will find a path with c colors with high probability after $O(e^c)$ trials, i.e. in time $2^{O(c)}m$.

The above algorithm is probabilistic, but it can be derandomized. A *c-perfect* family H of hash functions from the set of k colors to the set $\{1,\ldots,c\}$ is a family that has the property that for any set S of c colors there exists a function $h\in H$ such that h is one-to-one on S. As shown in [AYZ] (using [SS]), one can construct a *c-perfect* family H that contains $2^{O(c)}\log k$ functions, and each function can be evaluated in constant time on each element. Using the functions of such a c-perfect family in conjunction with the straightforward algorithm on the recolored DAG yields a deterministic algorithm that finds a path of the DAG that covers c colors (if there is such a path) in time $2^{O(c)}m\log k$. That is, Problem 3(2) can be solved in randomized $2^{O(c)}m$ time or deterministic $2^{O(c)}m\log k$ time.

4. EXPERIMENTS

We applied the algorithms to the test generation for the Personal HandyPhone System (PHS) and 5ESS Intelligent Network Application Protocol (INAP).

We model PHS by three EFSM's with 17 variables. The reachability graphs are DAG's. We test each machine separately using the longest path and the greedy heuristic procedures of Section 2.2. The results are similar and we report here those from the greedy heuristic. In the table, states and transitions refer to the number of control states and transitions in the EFSM, and edges and nodes concern the corresponding reachability graphs. Each test starts from the initial state with all the variable values zero and ends back to the initial state. The number of tests in a complete test set is also listed in the table.

machine	states	transitions	nodes	edges	tests
M_1	5	33	5	33	21
M_2	14	132	196	893	58
M_3	12	106	473	2,056	31

Table 1. Complete Test Sets for PHS Machines

For Machine 1, the first test covers 9 transitions, the second test covers 5 transitions, and the remaining 19 tests cover

one additional transition each. For Machine 2, the first 7 tests cover 78 transitions, and the remaining 54 transitions are covered by 51 tests where each test covers no more than 3 transitions. For Machine 3, the first 5 tests cover 73 transitions, and the remaining 33 transitions are covered by 26 tests where each test covers no more than 3 transitions.

In parallel with our efforts, a test set has also been generated manually with a complete coverage and contains approximately 200 test sequences. The total number of our tests is 110. As a matter of fact, after first few tests, the test sequences from our algorithm are minimal and complete test sets for the remaining untested transitions.

The second application is an on-going effort of testing 5ESS INAP. We model the system by an EFSM, which interacts with two other environment machines. The coverage required is not all the transitions but the code in the specification in VFSM notation (Virtual Finite State Machine) [Wa, HLS]. The system is large and there is no way to generate a complete reachability graph. A probabilistic algorithm is applied to obtain a subgraph for verification and testing so that a covering set of paths will provide a complete test set for the system [HLS].

The INAP machine has 140 control states, 187 variables, and 541 lines of specification code to be covered by tests. The (probabilistically generated) reachability graph has 296,423 nodes and 298,244 edges. As can be easily observed the graph is very sparse and a lot of nodes have degree 2. A straightforward application of our procedure generate 83 tests which have a complete coverage. However, more practical constraints (not reflected in the formal specification) are yet to be incorporated so that all the test sequences are valid.

5. CONCLUSION

We have discussed several optimization problems from feature testing of communication protocols and showed that most of them are NP-hard. We have proposed practical solutions and reported experimental results on real systems.

We have discussed test generation problems. Often system engineers come up with a number of tests based on their experiences or imagination that provide a good fault coverage. Their concern is: there is too much redundancy and, as a result, there are too many tests. They want to delete some tests without sacrificing the fault coverage. In this case, we have *test selection* rather than construction problems. Specifically, we have a complete test set, supposedly with a lot of redundancy. We want to select a subset of tests so that they are complete and has a minimal number: **From a complete test set, select a complete subset of minimum cardinality.**

Obviously, it is the minimum Set Cover problem [Ka, GJ, p. 222] and is NP-hard. As mentioned in Section 2, we can use a greedy method by repeatedly choosing the test with a maximal number of uncovered colors until all the colors are covered. Similar to path construction problems, it is known that it is optimal within a log factor of the exact number of minimal complete tests and that one cannot do better in polynomial time in general.

While testing of FSM's and the control portions of protocols is a well studied problem, testing of EFSM's and the data portions of protocols is still at an early stage; the difficulties are from the state explosion due to the large number of combinations of variable values. For systems of the size of PHS, the problem is still manageable. However, for systems such as INAP it is impossible to generate a reachability graph. The following minimization procedure is intended to reduce the size of the reachability graph for test generation and system analysis. Given a reachability graph from an EFSM, some nodes (configurations) may be equivalent and we want to minimize it by collapsing equivalent nodes first before test generation. Furthermore, we want to construct a minimized system even without first constructing a reachability graph, which is often impractical. It is known that this on-line minimization can be achieved in time polynomial in the size of the minimized systems [LY1].

Communication systems usually have timers and testing of the temporal properties is necessary. However, timers have an infinite range of values and their behaviors are difficult to model by variables as in EFSM's. Still it is possible to reduce the problem to a finite domain [ACD] and polynomial time algorithms are obtained for the minimization and reachability analysis [YL2].

Communication systems often exhibit nondeterministic behaviors due to the invisible internal transitions and the presence of timers [Ho]. Testing of nondeterministic systems even only for FSM's seems to be hard [ACY].

Protocol entities often send/receive parameters to/from each other and the executions (predicates and actions of the transitions) may depend on the parameter values received. To properly model and analyze the system behaviors, we further extend the model of EFSM and study *parameterized extended finite state machines* instead.

REFERENCES

[ADLU] A. V. Aho, A. T. Dahbura, D. Lee, and M. U. Uyar, "An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours," *IEEE Trans. on Communication*, vol. 39, no. 11, pp. 1604-15, 1991.

[ACD] R. Alur, C. Courcoubetis, and D. Dill, "Model checking for real-timed systems," in *Proc. 5th IEEE Symp. on Logic in Computer Science*, pp. 414-425, 1990.

[ACY] R. Alur, C. Courcoubetis, and M. Yannakakis, "Distinguishing tests for nondeterministic and probabilistic machines", *Proc. 27th Ann. ACM Symp. on Theory of Computing*, pp. 363-372, 1995.

[ANSI] International standard ISO 8802-2, ANSI/IEEE std 802.2, 1989.

[AYZ] N. Alon, R. Yuster, U. Zwick, "Color-coding," *J. ACM*, vol. 42, no. 4, pp. 844-856, 1995.

[ALMSS] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, "Proof verification and intractability of approximation problems," *Proc. of the 33rd IEEE Symp. on Foundations of Computer Science*, pp. 14-23, 1992.

[BU] S. C. Boyd and H. Ural, "On the complexity of generating optimal test sequences," *IEEE Trans. on Software Engineering*, vol. 17, no. 9, pp. 976-978, 1991.

[CZ] S. T. Chanson and J. Zhu, "A unified approach to protocol test sequence generation", *Proc. INFOCOM*, pp. 106-14, 1993.

[CK] K.-T. Cheng and A. S. Krishnakumar, "Automatic functional test generation using the extended finite state machine model," *Proc. DAC*, pp. 1-6, 1993.

[Ch] T. S. Chow, "Testing software design modeled by finite-state machines," *IEEE Trans. on Software Engineering*, vol. SE-4, no. 3, pp. 178-87, 1978.

[CLR] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, The MIT Press, 1990.

[DSU] A. T. Dahbura, K. Sabnani, and M. U. Uyar, "Formal methods for generating protocol conformance test sequences," *Proc. IEEE*, vol. 78, no. 8, August 1990.

[FBKAG] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test selection based on finite state models," *IEEE Trans. on Software Eng.*, vol. 17, pp. 591-603, 1991.

[Fei] U. Feige, "A threshold of ln*n* for approximating set cover," *Proc. 28th Ann. ACM Symp. Theory of Computing*, pp. 314-318, 1996.

[GJ] M. R. Garey and D. S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[Go] G. Gonenc, "A Method for the design of fault detection experiments," *IEEE Trans. Computers*, vol. C-19, pp. 551-58, 1980.

[He] F. C. Hennie, "Fault detecting experiments for sequential circuits," *Proc. 5th Ann. Symp. Switching Circuit Theory and Logical Design*, pp. 95-110, 1964.

[Ho] G. J. Holzmann, *Design and Validation of Protocols*, Prentice-Hall, 1990.

[HU] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

[HLS] S. Huang, D. Lee, and M. Staskauskas, "Validation-based Test Sequence Generation for Networks of Extended Finite State Machines," *Proc. FORTE/PSTV*, North Holland, R. Gotzhein Ed. 1996.

[Jo] D. S. Johnson. "Approximation algorithms for combinatorial problems," *J. of Computer and System Sciences*, vol. 9, pp. 256--278, 1974.

[Ka] R. M. Karp, "Reducibility Among Combinatorial Problems," *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher (eds.), Plenum Press, pp. 85-103, 1972.

[KSNM] K. Katsuyama, F. Sato, T. Nakakawaji, and T. Mizuno, "Strategic testing environment with formal description techniques," *IEEE Trans. on Computers*, vol. 40, no.4, pp. 514-25, 1991.

[KL] L.-S. Koh and M. T. Liu, "Test path selection based on effective domains," *Proc. of ICNP*, pp. 64-71, 1994.

[Ko] Z. Kohavi, *Switching and Finite Automata Theory*, 2nd Ed., McGraw-Hill, 1978.

[La] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Einehart and Winston, New York, 1976.

[LSKP] D. Lee, K. K. Sabnani, D. M. Kristol, and S. Paul, "Conformance testing of protocols specified as communicating finite state machines - a guided random walk based approach," *IEEE Trans. on Communications*, vol. 44, no. 5, pp. 631-640, 1996.

[Lo1] L. Lovasz. "On the ratio of optimal integral and fractional covers," *Discrete Mathematics*, vol. 13, pp. 383--390, 1975.

[Lo2] L. Lovasz. *Combinatorial Problems and Exercises*, North Holland, 1979.

[LY1] D. Lee and M. Yannakakis, "On-line minimization of transition systems," *Proc. 24th Ann. ACM Symp. on Theory of Computing*, pp. 264-274, 1992.

[LY2] D. Lee and M. Yannakakis, "Principles and Methods of Testing Finite State Machines - a Survey," *The Proceedings of IEEE*, Vol. 84, No. 8, pp. 1089-1123, August 1996.

[LuY] C. Lund and M. Yannakakis, "On the Hardness of Approximating Minimization Problems," *J. ACM* 41(5), pp. 960-981, 1994.

[ML] R. E. Miller and G. M. Lundy, "Testing protocol implementations based on a formal specification," *Protocol Test Systems III*, North-Holland, pp. 289-304, 1990.

[MP1] R. E. Miller and S. Paul, "On Generating conformance test sequences for combined control and data of communication protocols," *IFIP Protocol Specification, Testing, and Verification*, XII, pp. 13-27, 1992.

[MP2] R. E. Miller and S. Paul, "On the generation of minimal length conformance tests for communication protocols", *IEEE/ACM Trans. on Networking*, Vol. 1, No. 1, Feb. 1993, pp 116-129.

[MP3] R. E. Miller and S. Paul, "Structural Analysis of Protocol Specifications and Generation of Maximal Fault Coverage Conformance Test Sequences", *IEEE/ACM Trans. on Networking*, Vol. 2, No. 5, October 1994, pp 457-470.

[Mon] B. Monien, "How to find long paths efficiently," *Ann. Disc. Math.*, vol. 25, pp. 239-254, 1985.

[Mo] E. F. Moore, "Gedanken-experiments on sequential machines," in *Automata Studies*, Annals of Mathematics Studies, Princeton University Press, no. 34, pp. 129-53, 1956.

[NT] S. Naito and M. Tsunoyama, "Fault detection for sequential machines by transitions tours," *Proc. IEEE Fault Tolerant Comput. Symp.*, IEEE Computer

Society Press, pp. 238-43, 1981.

[PY] C. H. Papadimitriou, M. Yannakakis, "Optimization, Approximation and Complexity Classes," *J. Comp. Sys. Sc.* 43(3), 425-440, 1991.

[SD] K. K. Sabnani and A. T. Dahbura, "A protocol test generation procedure," *Computer Networks and ISDN Systems,* vol. 15, no. 4, pp. 285-97, 1988.

[SL] D. P. Sidhu and T.-K. Leung, "Formal methods for protocol testing: a detailed study," *IEEE Trans. Soft. Eng.*, vol. 15, no. 4, pp. 413-26, April 1989.

[SS] J. P. Schmidt and A. Siegel, "The spatial complexity of oblivious *k*-probe hash functions," *SIAM J. Comput.*, vol. 19, no. 5, pp. 775-786, 1990.

[Va] M. P. Vasilevskii, "Failure diagnosis of automata," *Kibernetika*, no. 4, pp. 98-108, 1973.

[Wa] F. Wagner, "VFSM executable specification," *Proc. CompEuro,* 1992.

[WL2] C.-J. Wang and M. T. Liu, "Generating test cases for EFSM with given fault models," *Proc. INFOCOM,* pp. 774-781, 1993.

[YJ] D. M. Yellin and C. S. Jutla, "Finding extremal sets in less than quadratic time," *Inf. Proc. Let.*, vol. 48, pp. 29-34, 1993.

[YL1] M. Yannakakis and D. Lee, "Testing finite state machines: fault detection," *J. of Computer and System Sciences, Vol. 50, No. 2, pp. 209-227, 1995.*

[YL2] M. Yannakakis and D. Lee, "An efficient algorithm for minimizing real-time transition systems," *Proc. CAV,* pp. 210-224, 1993.