(Computer Networks: The Int'l Journal of Comp. and Telecomm. Networking, 31(18):1967-1998, Sep 1999)

Testing Protocols Modeled as FSMs with Timing Parameters *

M. Ümit Uyar [a,1] Mariusz A. Fecko [b] Adarshpal S. Sethi [b] Paul D. Amer [b]

[a] *Electrical Engineering Department*
The City College of the City University of New York, NY

[b] *Computer and Information Sciences Department*
University of Delaware, Newark, DE

Abstract

An optimization method is introduced for generating minimum-length test sequences taking into account timing constraints for FSM models of communication protocols. Due to active timers in many of today's protocols, the number of consecutive self-loops that can be traversed in a given state before a timeout occurs is limited. A test sequence that does not consider timing constraints will likely be unrealizable in a test laboratory, thereby potentially resulting in the incorrect failing of valid implementations (or, vice versa). The solution uses a series of augmentations for a protocol's directed graph representation. The resulting test sequence is proven to be of minimum-length while not exceeding the tolerable limit of consecutive self-loops at each state. Although UIO sequences are used for state verification method, the results also are applicable to test generation that uses distinguishing or characterizing sequences.

Key words: conformance testing, test case generation, timing constraints, rural Chinese postman problem, protocol specification and testing.

* This work supported, in part, by the US Army Research Office Scientific Services Program administered by Battelle (DAAL03-91-C-0034), by the US Army Research Office (DAAL03-91-G-0086), and through collaborative participation in the Advanced Telecommunications/Information Distribution Research Program (ATIRP) Consortium sponsored by the U.S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0002.

[1] Dr. Uyar performed this research while a Visiting Associate Professor at University of Delaware.

1 Introduction

Due to interoperability requirements of heterogeneous devices in a complex communications network, each component of such a network must be tested for conformance against its specification. Automated generation of conformance tests based on the formal descriptions of communication protocols has been an active research area [1]–[16]. These techniques, using a deterministic finite-state machine (FSM) model of a protocol specification, focus on the optimization of the test sequence length. If, however, there exist timing constraints imposed by a protocol's active timers and these constraints are not considered during test sequence generation, the generated test sequence may not be realizable in a test laboratory. This can result in inconclusive or wrong verdicts such as the incorrect failing of valid implementations (or passing non-conformant ones).

In this paper, a solution is given to optimize the test sequence length and cost under the constraint that an implementation under test (IUT) can remain only a limited amount of time in some states during testing, before a timer's expiration forces a state change. The solution augments original graph representation of the protocol FSM model and formulates a Rural Chinese Postman Problem solution [17] to generate a minimum-length tour. In the final test sequence generated, the number of consecutive self-loops never exceeds any state's specified limit.

UIO sequences [18] are used for state verification throughout the paper. However, the results presented also applicable to test generation that uses the distinguishing or characterizing sequences [6,19,20] as discussed in Section 6.1. Earlier results of this study, limited to verification sequences that are self-loops, are presented in [21]. This paper generalizes these earlier results to both self-loop and non-self-loop verification sequences.

Section 2 presents some practical motivation behind the optimization problem formulated in the paper. Two real protocols, Q.931 [22] and MIL-STD 188-220B [23], demonstrate real examples of protocols with self-loop timing constraints. Section 3 provides the background information for FSM models and test generation. It also discusses the practical restrictions imposed on test sequences due to the timers. Section 4 defines different classes for UIO sequences based on the combination of edges and self-loops that a UIO sequence may contain. Section 5 presents the formal definition of the optimization problem. Finally, a solution for this optimization problem is presented in Section 6. Existence proofs of a polynomial-time solution are given in the Appendix.

2 Motivation

During testing, traversing each state transition of an IUT requires a certain amount of time. A test sequence that traverses too many *self-loops* (a *self-loop* is a state transition that starts and ends at the same state) in a given state will not be realizable in a test laboratory if the time to traverse the self-loops exceeds a timer limit as defined by another transition originating in this state. In this case, a timeout will inadvertently trigger forcing the IUT into a different state, and thereby disrupting the test sequence before all of the self-loops are traversed. If these timers are not taken into consideration during the test generation, most tests will result in either an

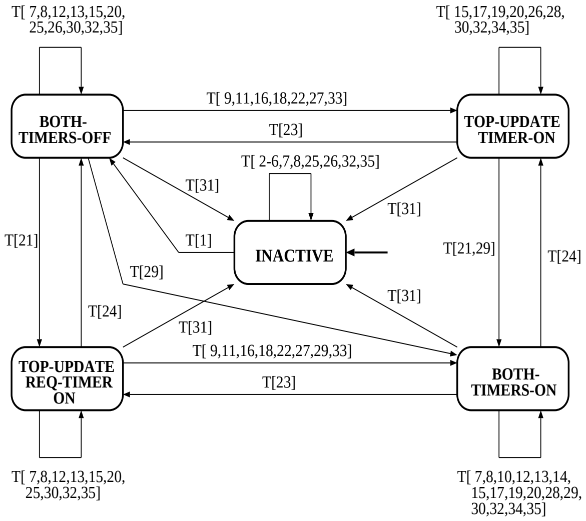


Fig. 1. Extended FSM for Topology Update module of MIL-STD 188-220B.

"inconclusive" verdict or, worse, a wrong verdict (i.e., failing the IUTs even when they meet the specification, or passing non-conformant IUTs). Clearly, this is not the goal of testing. Therefore, a properly generated test sequence must consider a protocol's timer constraints.

In general, the majority of tests defined for an IUT are classified into two categories: valid and inopportune tests [8,24]. Valid tests correspond to the "normal" or expected behavior of a protocol entity. Inopportune tests have inputs that are semantically and syntactically correct, but arrive at unexpected states (or, out of sequence). It is common practice that most inopportune messages are expected to be ignored by an IUT, which typically defines the edges representing them as self-loops with a null or warning output. (Although in the protocol specification only valid transitions are defined explicitly, the set of inopportune transitions can be derived.)

Examples of protocols that contain many self-loop transitions in their FSM models include ISDN Q.931 for supplementary voice services [22], MIL-STD 188-220B [23] for Combat Net Radio communication, and LAPD [25], the data link protocol for the ISDN's D channel.

In addition to the original self-loops of a specification model, extra self-loops are typically created when the test sequences use state verification techniques such as unique input/output (UIO) sequences [18], distinguishing sequences [19,20], or characterizing sequences [6,19,20].

Example 1: Timing constraints in MIL-STD 188-220B

The University of Delaware's Protocol Engineering Laboratory is developing test scripts to be used by the U.S. Army CECOM MIL-STD 188-220B Conformance Tester. Tests are being generated for both the Data Link and Intranet Layers. The tests are derived from an Estelle specification of the protocol. An extended FSM (i.e., FSM with memory) representing a portion of the Intranet Layer of 188-220B, called the Topology Update (TU), is shown in Figure 1 [26]. The equivalent FSM model of Topology Update has 10 states and 345 state transitions. In 8 of these states at least one timer is running in the implementation.

A timer's status (i.e., on or off) determines the behavior of the implementation. For example, when the topology information changes, the station is allowed to send a topology update message only if the *Topology_Update Timer* is not running. Otherwise, no message is sent. Based on this characteristic, the state names include the timer status in Figure 1.

There are 10 self-loop transitions defined for each of the states *TOP-UPDATE-REQ-TIMER-ON* and *TOP-UPDATE-TIMER-ON*, and 16 self-loops for state *BOTH-TIMERS-ON*. Depending on the timer expiration values, it may not be possible to execute all of the respective self-loop transitions during one visit to either state. Timing constraints due to the active timers must be taken into account to generate realizable test sequences for the Intranet Layer of 188-220B. Otherwise, valid implementations will fail the test sequence, which is not what the tester desires.

Example 2: ISDN Q.931

The portion of the Q.931 protocol that defines ISDN's basic voice services specifies 12 states and 16 different inputs for the user side. In the specification, there are 86 "normal" state transitions and 106 "inopportune" message transitions. Each inopportune transition is modeled as a self-loop with a null output.

In a test laboratory, an inopportune transition is tested by supplying its input to the IUT, and observing that the IUT does not generate any output. Usually, a timer is run by the tester to make sure that no output is generated. Then, to verify that the state of the IUT did not change, a *STATUS_INQUIRY* input is applied to the IUT, which generates an output called *STATUS*. The input of *STATUS_INQUIRY* and its output *STATUS* are self-loop transitions defined for each state.

Therefore, in Q.931, each state has an average of 9 inopportune transitions, which requires the traversal of 18 self-loop transitions during testing. The total ratio of self-loops to nonself-loop transitions is approximately 3 to 1 in the final test sequence. This ratio is even larger for the Q.931 supplementary voice services. LAPD, the ISDN data link layer protocol, demonstrates a similar characteristic: a high ratio of self-loop versus non-self-loop transitions.

A Q.931 implementation has several active timers that are running in certain states. For example, when an IUT moves from state *Null* to *Call Initiated*, a timer labeled as *T303* is started. When testing inopportune transitions in state *Call Initiated*, a tester has to consider a limited amount of time that can be used for inopportune tests before the timer expires. Other examples of timers in Q.931 are: timer *T304* running in state *Overlap Sending*, and timer *T310* in state *Outgoing Call Proceeding*.

3 Preliminaries and practical restrictions on test sequences

A *protocol* can be specified as a deterministic FSM [3,10,20], which can be represented by a directed graph $G = (V, E)$. The set $V = \{v_1, \ldots, v_n\}$ of vertices correspond to the set of states S of the FSM. A directed edge from v_i to v_j with label $L_k = a_l/o_m$, and the *cost* to realize the edge during testing, corresponds to a state transition in the FSM from s_i to s_j by applying input a_l and observing output o_m. If the start and the end vertices of an edge are the same (i.e., $v_i = v_j$), the edge is called a *self-loop*. The *indegree* and *outdegree* of a vertex are the number of edges coming toward and directed away from it, respectively. If the indegree and outdegree of each vertex are equal, the graph is said to be *symmetric*.

A positive integer is associated with each edge (v_i, v_j) to represent the *cost* to realize the edge during testing. A cost usually corresponds to the difficulty to exercise the corresponding state transition. Also, a non-negative integer representing an edge's *capacity* can be associated with each edge. The capacity is the maximum number of units of network flow that can be put on the edge [27] (i.e., the maximum number of times that this edge can be replicated, as discussed in Section 5.1).

A *tour* is a sequence of consecutive edges that starts and ends at the same vertex. An *Euler tour* is a tour that contains every edge of G exactly once. The so-called *Chinese Postman Problem* is defined as finding a minimum-cost tour of G that traverses every edge at least once [28]. The *Rural (Chinese) Postman Problem* is finding a (minimum-cost) tour for a subset of edges in G [17].

During conformance testing of a protocol implementation, the IUT is viewed as a *black box*, where only the inputs applied to the IUT and the outputs generated by the IUT can be observed, respectively. An IUT *conforms* to its specification if all state transitions defined in the specification are tested successfully. To test a single transition defined from state v_i to v_j, the following steps are needed:

- bring the IUT into state v_i;
- apply the required input and compare the output(s) generated with those defined by the specification;
- verify that the new state of the IUT is v_j by applying a state verification sequence.

As the last step of the above single transition test, the unique input-output (UIO) sequences [18] technique (see Section 4) is used throughout the paper. A UIO sequence of a state s_i, denoted $UIO(s_i)$, is a specified input/output sequence with the originating state s_i such that there is no $s_j \neq s_i$ for which $UIO(s_i)$ is a specified input/output sequence for s_i [18]. UIO sequences have been widely used in practice in testing communications protocols and devices, for example, ISDN systems and PBXs [29,30]. Section 6.1 presents a discussion on how to utilize other state verification techniques such as the distinguishing sequences [19,20] and characterizing (or W) sequences [6,19,20] for the results presented here.

Aho et al. introduced an optimization for the test sequence length (and cost) using UIO sequences [16]. Shen et al. [10], Ural and Lu [13] presented optimization methods using multiple UIO sequences for a given state. By taking advantage of repeated edge subsequences in a test sequence, heuristics to overlap the subsequences and further shorten the final test sequence are proposed by Chen et al. [14,15], Yang and Ural [9], and Miller and Paul [4,5].

All of these methods emphasize optimizing the test sequence length and its cost, without considering any restrictions on the order in which the tests can be applied to an IUT. One important restriction is due to timers that may be active in a given state. During testing, to realize a state transition takes a certain amount of time. A test sequence that traverses many consecutive self-loops in a state where a timer is running may not be realizable in a test laboratory. In this case, a timeout may disrupt the test sequence and move the IUT into a different state before all of the consecutive self-loops are exercised. This interruption increases the testing cost since a new setup must be prepared after each such break in the test sequence. Therefore, an optimization technique for generating realizable tests must consider the additional restriction that there is a limit on the number of self-loop transitions traversed consecutively.

This paper presents minimum-cost test sequence generation under the constraint that the number of consecutive self-loops that can be traversed during a visit to a given state is limited. In most cases, this test sequence will be longer than one without the constraint since limiting the number of self-loop traversals may require additional visits to a state which otherwise would have been unnecessary. In the test suite considered in this paper, valid and inopportune tests are handled together. This implies that the generated test sequence will test all self-loops of the protocol along with valid non-self-loop transitions.

Another choice for modeling the specifications with timing constraints could be timed automata [31,32]. However, the research on timed automata mainly concentrates on model checking rather than test generation. Hence, the existing literature on timed automata does not provide any extra help to obtain an efficient solution for the timing constraint problem investigated in this paper. Choosing the traditional FSM model over the timed automata, this paper presents a polynomial time algorithm for complex real-life protocols, such as 188-220 for combat network radios.

4 Classes of UIO sequences

A UIO sequence of a state v_i, $UIO(v_i)$, may contain both self-loop and non-self-loop edges. In general, $UIO(v_i)$ can be viewed as a concatenation of a number of (some possibly empty) subsequences (each subsequence by itself may or may not constitute a UIO sequence):

$$\begin{aligned} UIO(v_i) = & \, uio\_part(v_i, v_i) \cdot uio\_part(v_i, v_{j_0}) \cdot uio\_part(v_{j_0}, v_{j_0}) \cdot \ldots \\ & \cdot uio\_part(v_{j_{m-1}}, v_{j_m}) \cdot uio\_part(v_{j_m}, v_{j_m}) \end{aligned} \tag{1}$$

where $\cdot$ is a concatenation operator, a subsequence $uio\_part(v_{j_k}, v_{j_k})$ contains only self-loop edges of vertex v_{j_k}, and a subsequence $uio\_part(v_{j_{k-1}}, v_{j_k})$ is a path of non-self-loop edges starting at $v_{j_{k-1}}$ and ending at v_{j_k}. The length of $UIO(v_i)$, denoted as $|UIO(v_i)|$, is defined as the number of edges contained in $UIO(v_i)$. In (1), a UIO sequence $UIO(v_i)$ is said to contain each subsequence $uio\_part(v_{j_k}, v_{j_m})$, which is denoted as $uio\_part(v_{j_k}, v_{j_m}) \subset UIO(v_i)$.

Based on this definition, in general, there are three possible forms that $UIO(v_i)$ can have:

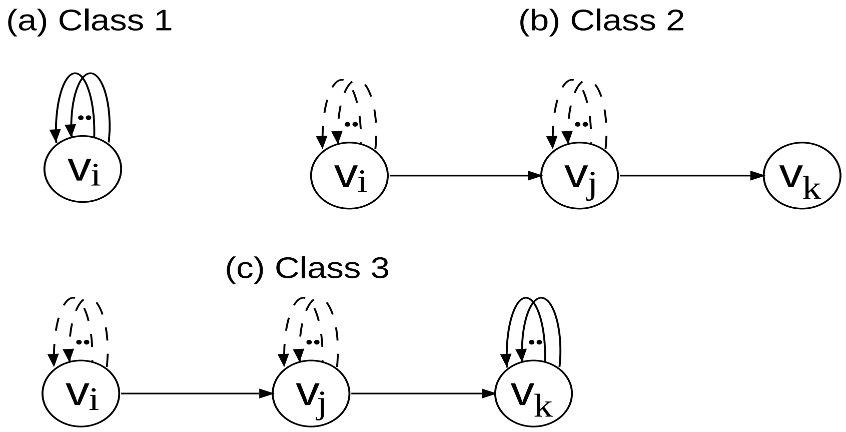


Fig. 2. Three general classes of UIO sequences. The figure shows only edges that belong to $UIO(v_i)$.

Class 1. $UIO(v_i) \equiv uio\_part(v_i, v_i)$

The $UIO(v_i)$ consists of only self-loops. Whenever an incoming edge of v_i or a self-loop edge of v_i is tested, applying the state verification sequence results in the FSM staying in state v_i. An example is given in Figure 2 (a). This class requires that the length of the UIO sequence satisfy $max\_self(v_i) \geq |UIO(v_i)|$, where $max\_self(v_i)$ is the maximum number of self-loops that can be traversed during one visit to v_i. If this condition does not hold, there is not enough time to verify the ending state of a transition before another transition fires due to timer expiration.

If all UIO sequences belong to Class 1, the test sequence (i.e., Chinese Postman tour) can be found in polynomial-time as described in [21].

Class 2. $UIO(v_i) \equiv uio\_part(v_i, v_i) \cdot uio\_part(v_i, v_{j_0}) \cdot uio\_part(v_{j_0}, v_{j_0}) \cdot ... \cdot uio\_part(v_{j_{m-1}}, v_{j_m})$

In this class, $UIO(v_i)$ may or may not start with a self-loop edge sequence, but it ends with a non-self-loop edge. See Figure 2 (b) for an example. In this case, every time a self-loop edge is tested, the UIO sequence moves the FSM out of v_i into v_k. Class 2 requires that vertex v_i satisfy $max\_self(v_i) \geq 1 + |uio\_part(v_i, v_i)|$. Moreover, each vertex v_{j_k} such that $uio\_part(v_{j_k}, v_{j_k}) \subset UIO(v_i)$ must satisfy $max\_self(v_{j_k}) \geq |uio\_part(v_{j_k}, v_{j_k})|$.

Class 3. $UIO(v_i) \equiv uio\_part(v_i, v_i) \cdot uio\_part(v_i, v_{j_0}) \cdot uio\_part(v_{j_0}, v_{j_0}) \cdot ... \cdot uio\_part(v_{j_k \neq i}, v_{j_k}) \cdot uio\_part(v_{j_k}, v_{j_k})$, where $|uio\_part(v_{j_k}, v_{j_k})| > 0$.

A Class 3 $UIO(v_i)$ contains non-self-loop edges and must end with one or more self-loop edges of state v_{j_k}. It may also contain self-loop edges at the beginning and in the middle. An example is in Figure 2 (c). Class 3 requires that vertex v_i satisfy $max\_self(v_i) \geq 1 + |uio\_part(v_i, v_i)|$. Each vertex v_{j_k} such that $uio\_part(v_{j_k}, v_{j_k}) \subset UIO(v_i)$ must satisfy $max\_self(v_{j_k}) \geq |uio\_part(v_{j_k}, v_{j_k})|$.

The existence of Class 3 adds extra complexity to the graph augmentations and the algorithms proposed in the paper. See Section 5.2.2 for an illustration.

Although finding UIO sequences is NP-hard for the general case, many researchers and practitioners report that the UIO sequences for most real-life protocols are short enough to be found in polynomial time [29,30]. If there is no UIO sequence found for a given state due to the timer constraints, either a different state verification method (such as the characterizing sequences) can be used, or the state verification can be skipped for that state since it is not possible to verify such a state. Also, note that it is relatively simple to add the timing constraints into the

UIO generation algorithms given in [18]. When any UIO sequence is considering the self-loops of a state v_i, the number of consecutive self-loops should be bounded by $max\_self(v_i)$. We believe that this additional timing constraint on finding the UIO sequences will significantly enhance their applicability to real-life protocols.

5 Problem formulation

Let the graph representing the FSM for a given protocol be $G(V, E)$. Let $\beta(v_i, v_j)$ and $\psi(v_i, v_j)$ be the capacity and cost of the edge $(v_i, v_j) \in E$, respectively, where $v_i, v_j \in V$.

Let us divide the vertices of G into three disjoint subsets corresponding to the three classes of UIO sequences used for the purpose of state verification:

$$\begin{aligned} V^{I} &\stackrel{def}{=} \{v_i \in V : UIO(v_i) \in Class\ 1\} \\ V^{II} &\stackrel{def}{=} \{v_i \in V : UIO(v_i) \in Class\ 2\} \\ V^{III} &\stackrel{def}{=} \{v_i \in V : UIO(v_i) \in Class\ 3\} \end{aligned}$$

Let us now divide the edges in E into two disjoint sets such that $E = E_{self} \cup E_{non-self}$:

- *self-loop transitions:* $E_{self} = \{(v_i, v_j) : v_i, v_j \in V \ \wedge \ v_i = v_j\}$
- *non-self-loop transitions:* $E_{non-self} = \{(v_i, v_j) : v_i, v_j \in V \ \wedge \ v_i \neq v_j\}$

Let $d_{out}(v_i)$ and $d_{in}(v_i)$ denote the out-degree and in-degree of vertex $v_i \in V$, respectively. Let the number of self-loops of vertex $v_i \in V$ be defined as:

$$d_{self}(v_i) \stackrel{def}{=} card\{(v_i, v_j) : v_j \in V \ \wedge \ (v_i, v_j) \in E_{self}\}$$

In a test sequence, at each visit to v_i, the maximum number of self-loops that can be traversed is $max\_self(v_i)$. As indicated in Section 3, attempting to remain in state v_i long enough to execute more than $max\_self(v_i)$ self-loops would result in disruption of a test sequence.

Let us consider an edge (v_i, v_j) incoming to a state v_j. Note that, if edge (v_i, v_j) does not start a timer, then all self-loops defined for state v_j can be tested after traversing edge (v_i, v_j). In this case, the problem of a test sequence disruption due to timeouts does not exist for state v_j. The technique presented in this paper prevents such a disruption when a state is reached through a transition which starts a timer.

Let $d_{min\_self}(v_i)$ be the minimum number of times a tour covering all edges of $E_{non-self} \cup E_{self}$ must include vertex $v_i \in V$. $d_{min\_self}(v_i)$ will be determined for each class of UIO sequences in Section 5.2.

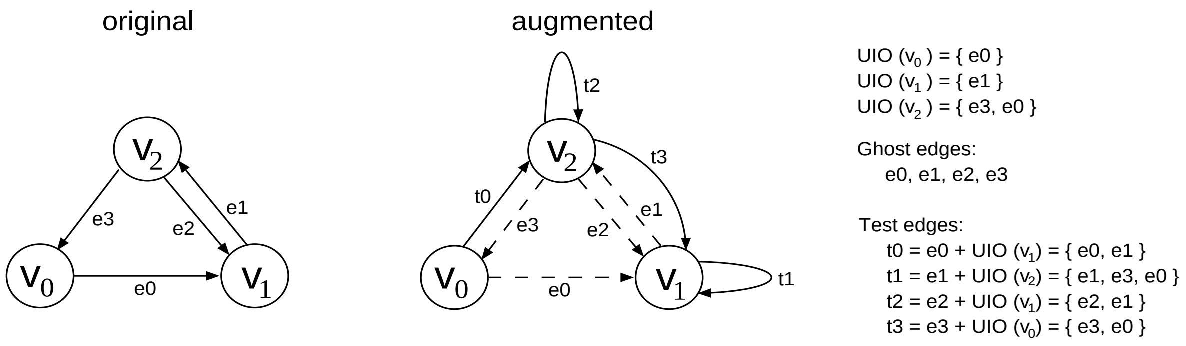


Fig. 3. Augmenting a graph with test and ghost edges.

5.1 Formulation of Rural Chinese Postman Problem

Let each edge $(v_i, v_j) \in E$ in G be replaced by a *test edge* $(v_i, v_k) \in E_{test}$ and a *ghost edge* $(v_i, v_j) \in E_{ghost}$. The test edge (v_i, v_k) is a concatenation of edge (v_i, v_j) and $UIO(v_j)$, where $UIO(v_j)$ ends at v_k. The cost of (v_i, v_k) is the sum of the costs of (v_i, v_j) and $UIO(v_j)$ (see Figure 3 for an example of augmenting a graph with test and ghost edges).

Our goal is to build a minimum-cost tour of G such that all edges in E_{test} (and some edges in E_{ghost}, if needed) are traversed with the constraint that each vertex v_i can only tolerate $max\_self(v_i)$ consecutive self-loop traversals.

Let g be a function of two arguments: an edge $e \in E$ and an integer k. The value of g is a set of $k \geq 0$ copies of its first argument $e \in E$. The function g represents the replications of an edge $e \in E$ in G. Let $\hat{E}_{test}$ be the set of all test edges that are a concatenation of a self-loop edge and a self-loop UIO sequence.

Let $G^{'}(V^{'}, E^{'})$ and its rural symmetric augmentation $G^{''}(V^{''}, E^{''})$ be the graphs satisfying the following conditions:

$$
\begin{aligned}
V^{'} \equiv V \ \wedge \ V^{''} &\equiv V \\
E^{'}_{test} &= E_{test} - \hat{E}_{test} \qquad (2)\\
E^{'} &= E_{ghost} \cup E^{'}_{test} \qquad (3)\\
E^{''} &= E^{'} \cup E_g, \ where \ E_g \stackrel{def}{=} \bigcup_{e_{rep} \in E_{ghost}} g(e_{rep}, f(e_{rep})) \qquad (4)\\
\forall v^{''}_i \in V^{''} d_{in}(v^{''}_i) &= d_{out}(v^{''}_i) \qquad (5)\\
\forall v^{''}_i \in V^{''} d_{in}(v^{''}_i) &\geq d_{min\_self}(v_i) \qquad (6)
\end{aligned}
$$

The function f is the maximum-flow minimum-cost function defining the rural symmetric augmentation of $G^{'}$, which will be discussed in Section 6.

(2) and (3) define $G^{'}$ as a graph containing all edges of G except for the test edges in $\hat{E}_{test}$ (edges in $\hat{E}_{test}$ will be added to a test sequence once it is found). In $G^{'}$, the imbalance of a node $v^{'} \in V^{'}$

is defined as the difference between the number of incoming and outgoing test edges of v'. This imbalance is eliminated by replicating (if needed) some of the incoming and/or outgoing ghost edges of v', for all $v' \in V'$. The resulting graph G'' is a rural symmetric augmentation of G' (as defined by (4) and (5)). By definition, in G'', the in-degree of any vertex $v_i'' \in V''$ is equal to its out-degree. Also, inequality (6) requires that the in-degree of any vertex $v_i'' \in V^I$ be greater or equal to the value defined by $d_{min\_self}(v_i)$, where v_i is the corresponding vertex in V. [2]

G' contains only test edges in $E_{test} - \hat{E}_{test}$. After obtaining a tour of G'', the test edges containing only self-loop transitions of a vertex v_i (i.e., the test edges in $\hat{E}_{test}$) will be added to the tour later at each visit to v_i.

Our goal is to build a Rural Chinese Postman tour in which the constraint set by (6) is satisfied for each vertex $v_i \in V$. A Rural Chinese Postman tour is a minimum-cost tour covering each transition $e \in E'_{test}$ exactly once, and each $e \in E_{ghost}$ zero or more times. Such a tour is equivalent to an Euler tour in a minimum cost rural symmetric augmentation G'' of G'. In other words, the objective is to obtain the graph G'' as the minimum-cost rural symmetric augmentation of the graph G'. Therefore, this goal is now reduced to finding the value of the function f in equation (4) above for all edges in E'.

5.2 Constraints on the number of visits to a state

Recall from Section 5 that $d_{min\_self}(v_i)$ is defined as the minimum number of times a tour covering all edges of $E_{non-self} \cup E_{self}$ must include vertex $v_i \in V$. Let us now derive the value of $d_{min\_self}(v_i)$ for vertices whose UIO sequence belongs to one of the three classes described in Section 4.

5.2.1 Class 1 UIO sequences

Testing an edge (v_j, v_i) in G involves traversing the edge followed by applying the UIO sequence of v_i. In a minimum-cost test sequence, an edge (v_j, v_i) may be traversed several times, but it will be tested only once, where (v_j, v_i) is followed by $UIO(v_i)$. In the case where edge (v_j, v_i) is tested, $max\_self(v_i) - |UIO(v_i)|$ self-loop traversals are left to be used for testing self-loops of v_i.

In general, to achieve minimum cost, we prefer a transition tour that does as much testing as possible when in a given state v_i. Therefore, the maximum number of self-loop transitions that can be tested during each visit requiring the state verification (after bringing the IUT to state v_i) is defined as

$$\Delta_1(v_i) = \lfloor \frac{max\_self(v_i) - |UIO(v_i)|}{1 + |UIO(v_i)|} \rfloor \tag{7}$$

[2] Note that, unless stated otherwise, v_i', v_i'' and v_i^* are used in this paper to denote the copies of a corresponding vertex $v_i \in V$ in graphs G', G'' and G^*, respectively.

Since there are exactly $d_{in}(v_i)$ non-self-loop edges ending at v_i, v_i must be visited at least $d_{in}(v_i)$ times, and for each visit $UIO(v_i)$ must be executed. The number of self-loop transitions that can be tested during all such visits to v_i is $d_{in}(v_i) * \Delta_1(v_i)$. Because the total number of self-loops of the vertex v_i that need to be tested is $d_{self}(v_i)$, all self-loop transitions can be tested during the required $d_{in}(v_i)$ visits to v_i only if

$$d_{self}(v_i) \leq (d_{in}(v_i) * \Delta_1(v_i))$$

In this case, $d_{min\_self}(v_i)$ is defined as

$$d_{min\_self}(v_i) \stackrel{def}{=} d_{in}(v_i) \tag{8}$$

which is sufficient for testing all edges with the ending state of v_i as well as all self-loops of v_i. Otherwise, the number of self-loop transitions remaining to be tested during subsequent visits to v_i is

$$d_{self}(v_i) - (d_{in}(v_i) * \Delta_1(v_i))$$

Vertex v_i may be the ending state of UIO sequences of vertices in $V^{II} \cup V^{III}$. Let $h(v_j, v_i)$ be the number of edges in E with the ending state of $v_j \in V^{II} \cup V^{III}$ such that $UIO(v_j)$ ends at vertex v_i. Formally, $h(v_j, v_i)$ is defined as:

$$h(v_j, v_i) \stackrel{def}{=} \begin{cases} d_{in}(v_j) + d_{self}(v_j) & \text{if } UIO(v_j) \text{ ends at } v_i \\ 0 & \text{otherwise} \end{cases}$$

Let $H(v_j, v_i)$ be the number of self-loop traversals of v_i included in $UIO(v_j)$. For $v_j \in V^{II}$, $H(v_j, v_i) = 0$; for $v_j \in V^{III}$, $H(v_j, v_i) = |uio\_part(v_i, v_i)|$, where $uio\_part(v_i, v_i) \subset UIO(v_j)$ (note that if $UIO(v_j)$ does not end at v_i, $H(v_j, v_i) \equiv 0$).

Therefore, the number of times a test sequence must visit v_i is given by

$$\sum_{v_j \in V^{II} \cup V^{III}} h(v_j, v_i)$$

each time permitting $max\_self(v_i) - H(v_j, v_i)$ self-loop traversals to be used for testing self-loops of v_i. The maximum number of self-loop transitions that can be tested during each such visit is defined as:

$$\Delta_2(v_j, v_i) = \lfloor \frac{max\_self(v_i) - H(v_j, v_i)}{1 + |UIO(v_i)|} \rfloor \tag{9}$$

In total, we can test $\Delta_2(v_i)$ self-loop edges of v_i as a result of executing UIO sequences of vertices in $V^{II} \cup V^{III}$ ending at v_i:

$$\Delta_2(v_i) = \sum_{v_j \in V^{II} \cup V^{III}} h(v_j, v_i) * \Delta_2(v_j, v_i) \qquad (10)$$

The following inequality holds if, after testing $d_{in}(v_i) * \Delta_1(v_i) + \Delta_2(v_i)$ self-loops of v_i, there are no remaining self-loops to be tested at v_i:

$$d_{self}(v_i) \leq (d_{in}(v_i) * \Delta_1(v_i)) + \Delta_2(v_i) \qquad (11)$$

In this case, all self-loop transitions can be tested during the required

$$d_{min\_self}(v_i) \stackrel{def}{=} d_{in}(v_i) + \sum_{v_j \in V^{II} \cup V^{III}} h(v_j, v_i) \qquad (12)$$

visits to v_i. Otherwise, extra visits to v_i are needed, as discussed below.

After each subsequent (i.e., extra) traversal of edges ending at v_i, it is possible to test $\Delta_3(v_i)$ self-loops:

$$\Delta_3(v_i) = \lfloor \frac{max\_self(v_i)}{1 + |UIO(v_i)|} \rfloor \qquad (13)$$

Note that (13) differs from (7) because no state verification of the transition entering state v_i is necessary.

In this case, $d_{min\_self}(v_i)$ is defined as

$$d_{min\_self}(v_i) \stackrel{def}{=} d_{in}(v_i) + \lceil \frac{d_{self}(v_i) - (d_{in}(v_i) * \Delta_1(v_i)) - \Delta_2(v_i)}{\Delta_3(v_i)} \rceil + \sum_{v_j \in V^{II} \cup V^{III}} h(v_j, v_i) \qquad (14)$$

which applies when condition (11) does not hold.

5.2.2 Class 2 and Class 3 UIO sequences

Let us now consider the minimum number of visits to vertices whose UIO sequences belong to Class 2 and Class 3 (i.e., $v_i \in V^{II} \cup V^{III}$).

If an edge incoming to $v_i \in V^{II} \cup V^{III}$ is being tested, $UIO(v_i)$ is applied. Since the IUT then moves to another state to complete $UIO(v_i)$, no self-loop edges of v_i can be tested during such a visit. Similar to the case of Class 1, there are

$$\sum_{v_j \in V^{II} \cup V^{III}} h(v_j, v_i)$$

visits to v_i resulting from testing edges that end at states whose UIO sequence ends at v_i. During each such visit, we can test $\Delta_2(v_j, v_i)$ self-loops of v_i:

$$\Delta_2(v_j, v_i) \stackrel{def}{=} \begin{cases} 1 \text{ if } max\_self(v_i) > H(v_j, v_i) + (1 + |uio\_part(v_i, v_i)|), \text{ where } uio\_part(v_i, v_i) \subset UIO(v_i) \\ 0 \text{ otherwise} \end{cases}$$

Intuitively, $\Delta_2(v_j, v_i) = 1$ means that after arriving at v_i and traversing $H(v_j, v_i)$ self-loops as part of $UIO(v_j)$, we can test one self-loop of v_i followed by $UIO(v_i)$. On the other hand, $\Delta_2(v_j, v_i) = 0$ implies that upon arriving at v_i there is no time to traverse a self-loop edge followed by self-loops of $UIO(v_i)$.

We can test total of $\Delta_2(v_i)$ self-loop edges of v_i, as defined in equation (10). After testing $\Delta_2(v_i)$ self-loops, there are no untested self-loops left only if:

$$d_{self}(v_i) \leq \Delta_2(v_i) \tag{15}$$

In this case, all self-loop transitions can be tested during the required

$$d_{min\_self}(v_i) \stackrel{def}{=} d_{in}(v_i) + \sum_{v_j \in V^{II} \cup V^{III}} h(v_j, v_i) \tag{16}$$

visits to v_i. If condition (15) does not hold, the test sequence must come back to v_i via incoming edges. The number of such visits to v_i is the number of remaining self-loop edges $d_{self}(v_i) - \Delta_2(v_i)$. Therefore, $d_{min\_self}(v_i)$ in this case is defined as:

$$d_{min\_self}(v_i) \stackrel{def}{=} d_{in}(v_i) + (d_{self}(v_i) - \Delta_2(v_i)) + \sum_{v_j \in V^{II} \cup V^{III}} h(v_j, v_i) \tag{17}$$

6 Minimum-cost solutions for constrained self-loop testing

We present the following solution to the problem of finding a symmetric G'' from G' while satisfying the constraint set in (6). Let $d_{in}^{E'_{test}}(v_i')$ and $d_{out}^{E'_{test}}(v_i')$ be the in-degree and the out-degree of v_i' based only on the number of test edges in E'_{test} incident on v_i', respectively.

The transformation applied to G' depends on the class of UIO sequence of its vertices. First, $G'(V', E')$ is converted to $G^*(V^*, E^*)$ by splitting each vertex $v_i' \in V^I$ satisfying predicate P_1

$$P_1(v_i) = (d_{min\_self}(v_i) > max(d_{in}^{E'_{test}}(v'_i), d_{out}^{E'_{test}}(v'_i))) \quad (18)$$

into the two vertices $v_i^{*(1)}$, $v_i^{*(2)} \in V^*$ (Figure 4).

Then, $v_i^{*(1)}$ is connected to $v_i^{*(2)}$ with a single ghost edge. The set of all edges connecting split vertices is defined as follows:

$$E_1^* \stackrel{def}{=} \bigcup_{v'_i \in V^I} \{(v_i^{*(1)}, v_i^{*(2)})\}$$

The test edges starting and ending at vertices in V^I are included in G^* as follows:

$$E_2^* \stackrel{def}{=} \{(v_i^{*(2)}, v_j^{*(1)}) : (v'_i, v'_j) \in E'_{test} \ \wedge \ v'_i, v'_j \in V^I\}$$

If $v'_i \in V^{II} \cup V^{III}$, the construction is different. Let $E_{test}^s(v'_i)$ and $E_{test}^{ns}(v'_i)$ be the sets of outgoing test edges of v'_i obtained from a self-loop edge starting at v'_i and a non-self-loop edge starting at v'_i, respectively.

The following predicate

$$P_2(v_i) = ((\exists v_k \in V) \ h(v_k, v_i) > 0 \ \wedge \ \Delta_2(v_k, v_i) = 0) \quad (19)$$

implies that the test edge whose UIO sequence starts at v'_k and ends at v'_i cannot be followed by any test edge whose first component is a self-loop of v'_i (i.e., the test edges in $E_{test}^s(v'_i)$). In this case, $UIO(v_k)$ will traverse some self-loops at v'_i (since $h(v_k, v_i) > 0$), but there is insufficient time to traverse any self-loops in an edge of $E_{test}^s(v'_i)$ (since $\Delta_2(v_k, v_i) = 0$).

Each $v'_i \in V^{II} \cup V^{III}$ satisfying (19) is split into two vertices $v_i^{*(1)}$, $v_i^{*(2)} \in V^*$ (Figure 5), and a single ghost edge $(v_i^{*(1)}, v_i^{*(2)})$ between them is added to the set E_3^*:

$$E_3^* \stackrel{def}{=} \bigcup_{v'_i \in V^{II} \cup V^{III}} \{(v_i^{*(1)}, v_i^{*(2)})\}$$

Afterwards, the incoming test edges of $v'_i \in V^{II} \cup V^{III}$ are added to G^* as follows:

$$E_4^* \stackrel{def}{=} \{(\bar{v}_j^*, \bar{v}_i^*) : (v'_j, v'_i) \in E'_{test}\}$$

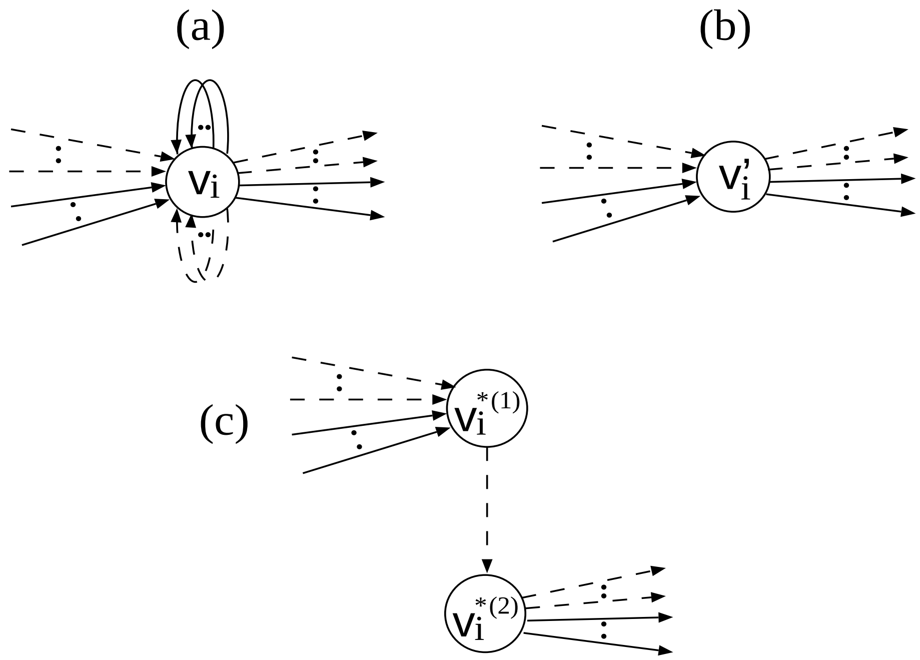


Fig. 4. Conversion of $v_i \in V^I$ in G (part (a)), to $v_i^{'}$ in $G^{'}$ (part (b)) and to $v_i^{*(1)}, v_i^{*(2)}$ in G^* (part (c)). Ghost edges appear in dashed lines.

where $\bar{v}_i^*$ is given by:

$$\bar{v}_i^* \stackrel{def}{=} \begin{cases} v_i^{*(1)} \text{ if } P_2(v_i) \text{ and } h(v_j, v_i) > 0 \ \wedge \ \Delta_2(v_j, v_i) = 1 \\ v_i^{*(2)} \text{ if } P_2(v_i) \text{ and } h(v_j, v_i) > 0 \ \wedge \ \Delta_2(v_j, v_i) = 0 \\ v_i^* \quad \text{if not } P_2(v_i) \end{cases}$$

The definition of $\bar{v}_j^*$ depends on the class of $UIO(v_j)$:

$$\bar{v}_j^* \stackrel{def}{=} \begin{cases} v_j^{*(1)} \text{ if } P_2(v_j) \text{ and } v_j^{'} \in V^{II} \cup V^{III} \text{ and } (v_j^{'}, v_i^{'}) \in E_{test}^{s}(v_j^{'}) \\ v_j^{*(2)} \text{ if } P_2(v_j) \text{ and } v_j^{'} \in V^{II} \cup V^{III} \text{ and } (v_j^{'}, v_i^{'}) \in E_{test}^{ns}(v_j^{'}) \\ v_j^* \quad \text{if not } P_2(v_j) \text{ and } v_j^{'} \in V^{II} \cup V^{III} \\ v_j^{*(2)} \text{ if } P_1(v_j) \text{ and } v_j^{'} \in V^I \\ v_j^* \quad \text{if not } P_1(v_j) \text{ and } v_j^{'} \in V^I \end{cases}$$

The test edges starting at $v_i \in V^{II} \cup V^{III}$ and ending at $v_j \in V^I$ are transformed in G^* in the following manner:

$$E_5^* \stackrel{def}{=} \{(v_i^{*(1)}, v_j^{*(1)}) : (v_i^{'}, v_j^{'}) \in E_{test}^{s}(v_i^{'})\} \cup \{(v_i^{*(2)}, v_j^{*(1)}) : (v_i^{'}, v_j^{'}) \in E_{test}^{ns}(v_i^{'})\}$$

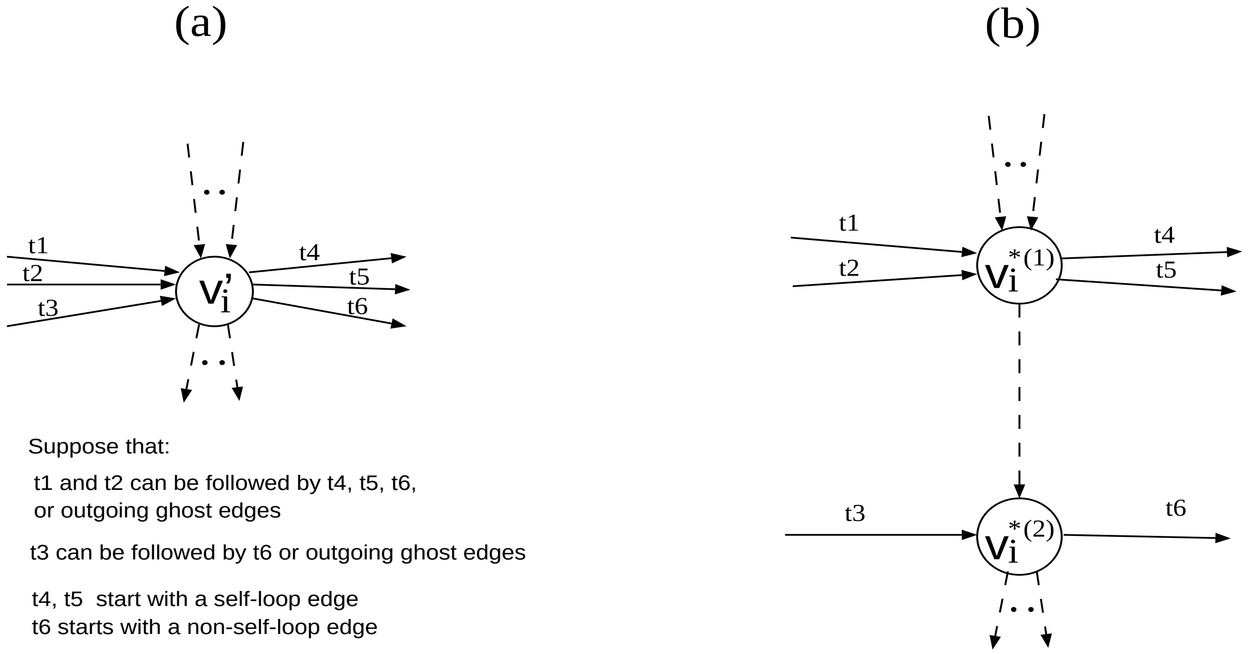


Fig. 5. Conversion of $v_i^{'} \in V^{II} \cup V^{III}$ in $G^{'}$ (part (a)) to $v_i^{*(1)}, v_i^{*(2)}$ in G^* (part (b)).

Finally, the ghost edges in $E^{'}$ are added to E^* as follows:

$$E_6^* \stackrel{def}{=} \{(v_i^{*(2)}, v_j^{*(1)}) : (v_i^{'}, v_j^{'}) \in E_{ghost}^{'}\}$$

Note that if any $v_k^{'}$ is not split, then the above definitions of the sets E_2^*, E_5^*, and E_6^* remain valid after substituting v_k^* for $v_k^{*(1)}$ and $v_k^{*(2)}$.

The last step of the conversion is the definition of test edges in G^* and the addition of the source and sink vertices (s and t, respectively):

$$\begin{aligned} E_{test}^* &\stackrel{def}{=} E_2^* \cup E_4^* \cup E_5^* \ \wedge \ E_{ghost}^* \stackrel{def}{=} E_1^* \cup E_3^* \cup E_6^* \\ E^* &\stackrel{def}{=} E_{test}^* \cup E_{ghost}^* \ \cup \bigcup_{v_i^{'} \in V^{'}} \{(s, v_i^{*(1)}), (s, v_i^{*(2)}), (v_i^{*(1)}, t), (v_i^{*(2)}, t)\} \end{aligned} \tag{20}$$

The problem of finding the minimum-cost rural symmetric augmentation of $G^{'}$ as $G^{''}$ then can be reduced to finding the integer function $f : E \to N$ whose value $f(e^* \in E^*)$ determines the number of times the corresponding edge $(e^{'} \in E^{'})$ needs to be included in the graph $G^{''}$ to make $G^{'}$ symmetric (Figure 4).

Aho et al. [16] presented an efficient solution to this problem for FSMs with either a self-loop property or a reset capability (see Appendix A for their definitions). Let us now apply a similar approach to the problem of minimizing the test sequence with the above self-loop repetition constraints, which is to maximize the flow on graph G^* with minimum cost. The set of test edges in G^* (which are all non-self-loop edges) will be referred to as E^*_{test} (defined in (20)). Edges incident to the source s and sink t in G^* are assigned capacity β as follows:

$$v_i \in V^I \Rightarrow \begin{cases} \beta(s, v_i^{*(1)}) &= \max(0, d_{in}^{E^*_{test}}(v_i^{*(1)}) - d_{min\_self}(v_i)) \\ \beta(s, v_i^{*(2)}) &= \max(0, d_{min\_self}(v_i) - d_{out}^{E^*_{test}}(v_i^{*(2)})) \\ \beta(v_i^{*(1)}, t) &= \max(0, d_{min\_self}(v_i) - d_{in}^{E^*_{test}}(v_i^{*(1)})) \\ \beta(v_i^{*(2)}, t) &= \max(0, d_{out}^{E^*_{test}}(v_i^{*(2)}) - d_{min\_self}(v_i)) \end{cases} \tag{21}$$

$$v_i \in V^{II} \cup V^{III} \Rightarrow \begin{cases} \beta(s, v_i^{*(1)}) &= \max(0, d_{in}^{E^*_{test}}(v_i^{*(1)}) - d_{out}^{E^*_{test}}(v_i^{*(1)})) \\ \beta(s, v_i^{*(2)}) &= \max(0, d_{in}^{E^*_{test}}(v_i^{*(2)}) - d_{out}^{E^*_{test}}(v_i^{*(2)})) \\ \beta(v_i^{*(1)}, t) &= \max(0, d_{out}^{E^*_{test}}(v_i^{*(1)}) - d_{in}^{E^*_{test}}(v_i^{*(1)})) \\ \beta(v_i^{*(2)}, t) &= \max(0, d_{out}^{E^*_{test}}(v_i^{*(2)}) - d_{in}^{E^*_{test}}(v_i^{*(2)})) \end{cases} \tag{22}$$

The cost ψ is zero for all edges starting at the source, all edges ending at the sink, and the edges connecting split vertices. The capacity for the edges connecting split vertices is infinite. Each of the remaining edges in E^* (i.e., the edges corresponding to original edges in E') has infinite capacity with the cost of the original edge in E'.

f is the maximum-flow minimum-cost function defined on the graph $G^*(V^*, E^*)$ that saturates all edges incident to s and t, i.e., $(\forall v_i^* \in V^* - \{s,t\})\ \beta(s, v_i^*) = f(s, v_i^*)$ and $\beta(v_i^*, t) = f(v_i^*, t)$. The function f satisfying this condition exists iff

$$\sum_{v_i^* \in V^* - \{s,t\}} \beta(s, v_i^*) = \sum_{v_i^* \in V^* - \{s,t\}} \beta(v_i^*, t)$$

which holds true for capacity assignments defined by (21) and (22) [27].

Let χ be an integer function whose value $\chi(v_i, v_j)$ is the number of times an edge (v_i, v_j) is included in G''. χ is defined as follows:

$$\chi(v_i, v_j) \stackrel{def}{=} \begin{cases} 1 + f(v_i, v_j) & \text{if } (v_i, v_j) \in E^*_{test} \\ f(v_i, v_j) & \text{if } (v_i, v_j) \in E^*_{ghost} - E^*_1 \\ d_{min\_self}(v_i) + f(v_i, v_j) & \text{if } (v_i, v_j) \in E^*_1 \end{cases}$$

It can be seen from Figure 4, that, for each vertex $v_i' \in V^I$ split by the algorithm (i.e., each v_i' for which condition (18) holds) the values of flow into $v_i^{*(1)}$ and out of $v_i^{*(2)}$ satisfy the following conditions:

$$\sum_{v_j^* \in V^* - \{s,t\}} f(v_j^*, v_i^{*(1)}) \geq d_{min\_self}(v_i) - d_{in}^{E_{test}^*}(v_i^{'}) \tag{23}$$

$$\sum_{v_j^* \in V^* - \{s,t\}} f(v_i^{*(2)}, v_j^*) \geq d_{min\_self}(v_i) - d_{out}^{E_{test}^*}(v_i^{'}) \tag{24}$$

From the definition of function χ, and equations (23) and (24), we obtain:

$$\sum_{v_j^* \in V^* - \{s,t\}} \chi(v_j^*, v_i^{*(1)}) = \sum_{v_j^* \in V^* - \{s,t\}} f(v_j^*, v_i^{*(1)}) + d_{in}^{E_{test}^*}(v_i^{'}) \ \geq \ d_{min\_self}(v_i)$$

$$\sum_{v_j^* \in V^* - \{s,t\})} \chi(v_i^{*(2)}, v_j^*) = \sum_{v_j^* \in V^* - \{s,t\}} f(v_i^{*(2)}, v_j^*) + d_{out}^{E_{test}^*}(v_i^{'}) \ \geq \ d_{min\_self}(v_i)$$

$$\sum_{v_j^* \in V^* - \{s,t\}} \chi(v_j^*, v_i^{*(1)}) = \sum_{v_j^* \in V^* - \{s,t\})} \chi(v_i^{*(2)}, v_j^*)$$

Therefore, each $v_i^{'} \in V^I$ will have at least $d_{min\_self}(v_i)$ outgoing edges after replication.

Vertices $v_i^{'} \in V^I$ that are not split by the algorithm (i.e., each $v_i^{'}$ for which condition (18) does not hold) will have at least $max(d_{in}^{E_{test}^*}(v_i), d_{out}^{E_{test}^*}(v_i)) \geq d_{min\_self}(v_i)$ outgoing edges after replication. Then in an Euler tour of $G^{''}$, each vertex $v_i^{''} \in V^I$ will be visited at least $d_{min\_self}(v_i)$ times.

Let G_s^* be the symmetric augmentation of G^* defined by the function χ. Note that G^* is identical to $G^{'}$ if all split vertices of $v_i^{*(1)}$ and $v_i^{*(2)}$ are merged into a single vertex $v_i^{'}$, and the ghost edges $(v_i^{*(1)}, v_i^{*(2)})$ are eliminated. In this case, and Euler tour T_s^* of G_s^* can be converted to an Euler tour $T^{''}$ of $G^{''}$ by skipping the $(v_i^{*(1)}, v_i^{*(2)})$ ghost edges in T_s^*. Therefore, $T^{''}$ can be obtained by replacing each occurrence of edges

$$(\bar{v}_j^*, v_i^{*(1)}), \ (v_i^{*(1)}, v_i^{*(2)}), \ (v_i^{*(2)}, \bar{v}_k^*)$$

in T_s^* ($\bar{v}_j^*$ denotes either $v_j^*, v_j^{*(1)}$, or $v_j^{*(2)}$ in G^*, whichever applies for $v_j^{'}$ in $G^{'}$) with the corresponding sequence of edges

$$(\bar{v}_j^*, v_i^{*(1)}), \ (v_i^{*(2)}, \bar{v}_k^*)$$

in $T^{''}$. Finally, all vertices $v_i^{*(1)}, v_i^{*(2)}$ and v_i^* in T_s^* should be replaced with the corresponding $v_i^{''}$ in $T^{''}$. It is clear that $\psi(T^{''}) = \psi(T_s^*)$.

Each test edge incident on $v_i^{'} \in V^I \cup V^{III}$ will be included in the tour such that incoming test edges of $v_i^{*(1)}$ may be followed by any outgoing test edge in $E_{test}^s(v_i^{'}) \cup E_{test}^{ns}(v_i^{'})$ or any outgoing ghost edge of $v_i^{'}$. On the other hand, the incoming test edges of $v_i^{*(2)}$ will be followed only by the outgoing test edges in $E_{test}^{ns}(v_i^{'})$ or the outgoing ghost edges of $v_i^{*(2)}$. Therefore, $T^{''}$ will not be disrupted by timeouts when implemented as a test sequence.

For G_s^* to have an Euler tour, G_s^* must be strongly-connected. Aho et al. [16] showed that the sufficient condition for strong-connectivity of G_s^*, where G_s^* includes all edges in $E_{test}^* \cup E_1^*$, is that the edge-induced subgraph $G[E_{test}^* \cup E_1^*]$ should be a weakly-connected spanning subgraph of G^*. It can be proven that, if the FSM has a reset capability or a self-loop property, $G[E_{test}^* \cup E_1^*]$ is a weakly-connected spanning subgraph of G^* (see Appendix A).

Example 3: Consider an FSM whose all UIO sequences belong to Class 1, as shown in Figure 6 (page 29). Suppose that vertices v_0, v_2 and v_3 of the FSM can tolerate at most three, and vertex v_1 at most two self-loop transitions during each visit. Let transitions $e10$ and $e11$ correspond to timeouts. After either $e10$ or $e11$ is triggered, the FSM is brought into state v_3. UIO sequences and the values of max_self, $|UIO|$ and d_{min_self} are:

Vertex	*UIO sequence*	max_self	$\lvert UIO \rvert$	d_{min_self}
v_0	e0	3	1	2
v_1	e2	2	1	3
v_2	e6,e7	3	2	4
v_3	e9	3	1	2

As described earlier, the original edges are replaced by the ghost edges and test edges (note that the ghost edges still remain in the graph). Since all UIO sequences are self-loops, the starting and ending state of a ghost edge and the corresponding test edge are the same, as shown in Figure 6 and Table 1.

The rural Chinese postman method [16], when applied to the graph without self-loop repetition constraint, results in the test sequence

$$\overbrace{e0, \mathbf{e0}}^{t0}, \overbrace{e1, \mathbf{e2}}^{t1}, \overbrace{e2, \mathbf{e2}}^{t2}, \overbrace{e10, \mathbf{e9}}^{t10}, \overbrace{e9, \mathbf{e9}}^{t9}, \overbrace{e12, \mathbf{e0}}^{t12}, e1, \overbrace{e3, \mathbf{e2}}^{t3}, \overbrace{e4, \mathbf{e6}, \mathbf{e7}}^{t4},$$
$$\overbrace{e6, \mathbf{e6}, \mathbf{e7}}^{t6}, \overbrace{e11, \mathbf{e9}}^{t11}, e12, e1, e4, \overbrace{e7, \mathbf{e6}, \mathbf{e7}}^{t7}, \overbrace{e8, \mathbf{e6}, \mathbf{e7}}^{t8}, \overbrace{e5, \mathbf{e0}}^{t5} \tag{25}$$

containing 34 edges (the edges used for the purpose of UIO state verification appear in bold).

As can be seen from the beginning part of the above test sequence

$$e0, \mathbf{e0}, e1, \mathbf{e2}, e2, \mathbf{e2}, e10, ...$$

it is required that, after $e1$ is traversed, the IUT should stay in state v_1 for a time that allows at least three $e2$ self-loop traversals. However, this part of the test sequence is not realizable in a test laboratory because the timeout edge $e10$ will be triggered after the second consecutive self-loop traversal (i.e., $max\_self(v_1) = 2$). The IUT will move into v_3 and further input/output exchanges are likely to fail even correct IUTs.

Table 1
Test and ghost edges for the graph of Figure 6 (a)

Test edge	*Start vertex*	*End vertex*	*Edges included*
t0	v_0	v_0	e0, **e0**
t1	v_0	v_1	e1, **e2**
t2	v_1	v_1	e2, **e2**
t3	v_1	v_1	e3, **e2**
t4	v_1	v_2	e4, **e6,e7**
t5	v_2	v_0	e5, **e0**
t6	v_2	v_2	e6, **e6,e7**
t7	v_2	v_2	e7, **e6,e7**
t8	v_2	v_2	e8, **e6, e7**
t9	v_3	v_3	e9, **e9**
t10	v_1	v_3	e10, **e9**
t11	v_2	v_3	e11, **e9**
t12	v_3	v_0	e12, **e0**

Similarly, consider the following part of the test sequence (25):

$$..., e4, \mathbf{e6}, \mathbf{e7}, e6, \mathbf{e6}, \mathbf{e7}, e11, ...$$

After $e4$ is traversed, the IUT should stay in state v_2 for a time necessary for five self-loop traversals, which will be impossible because $max\_self(v_2) = 3$. This subsequence can only be run in a laboratory as

$$..., e4, \mathbf{e6}, \mathbf{e7}, e6, e11, ...$$

where after three consecutive self-loops transitions **e6**, **e7**, e6, the sequence will prematurely take the IUT into state v_3. Again, the test sequence is disrupted by the $e11$ timeout event.

To address the problem of test sequence disruption due to timeouts, the graph of Figure 6 (a) is converted by the method described in Section 6 to the graph shown in Figure 6 (b). The vertices for which condition (18) holds, which are v_1 and v_2, are split and then connected by a single ghost edge.

Considering the self-loop constraint, the test sequence for the graph of Figure 6 (b) is obtained as

$$
\begin{aligned}
&\overbrace{e0, \mathbf{e0}}^{t0}, \overbrace{e1, \mathbf{e2}}^{t1}, \overbrace{e10, \mathbf{e9}}^{t10}, \overbrace{e9, \mathbf{e9}}^{t9}, \overbrace{e12, \mathbf{e0}}^{t12}, e1, \overbrace{e2, \mathbf{e2}}^{t2}, \overbrace{e4, \mathbf{e6}, \mathbf{e7}}^{t4}, \overbrace{e11, \mathbf{e9}}^{t11}, e12, \\
&e1, \overbrace{e3, \mathbf{e2}}^{t3}, e4, \overbrace{e6, \mathbf{e6}, \mathbf{e7}}^{t6}, \overbrace{e5, \mathbf{e0}}^{t5}, e1, e4, \overbrace{e7, \mathbf{e6}, \mathbf{e7}}^{t7}, e5, e1, e4, \overbrace{e8, \mathbf{e6}, \mathbf{e7}}^{t8}, e5
\end{aligned}
\tag{26}
$$

containing 40 edges. Although the test sequence in Figure 6 (b) is longer than that of Figure 6 (a), it is minimum-length given the self-loop constraint. During each visit to vertices v_0, v_1, v_2 and v_3, the number of consecutive self-loop edges traversed is less than or equal to the maximum allowed number of self-loop traversals. Therefore, this test sequence is realizable in a test laboratory.

Example 4: Consider an FSM whose UIO sequences belong to all three possible classes (Figure 7, page 29). Suppose that the maximum tolerable number of consecutive self-loop traversals is one for vertex v_0, two for v_1, and three for vertices v_2 and v_3. Let $e6$ and $e7$ be timeout transitions. When either of them is triggered, an IUT moves into state v_3. UIO sequences and the values of max_self and d_{min_self} are:

Vertex	*UIO sequence*	*Class of UIO sequence*	max_self	d_{min_self}
v_0	e0, e2	Class 3	1	4
v_1	e1, e5	Class 2	2	9
v_2	e12	Class 1	3	5
v_3	e13	Class 1	3	2

After replacing the original transitions by the ghost and test edges, we obtain the set of test edges for the graph of Figure 7 (a) as shown in Table 2.

Testing of $e8$ involves traversing one self-loop of v_1 (i.e., $e2$) as part of UIO sequence of v_0. Since $UIO(v_1)$ starts with a self-loop (i.e., $e1$) and $max\_self(v_1) = 2$, no self-loops of v_1 can be tested immediately after testing $e8$. This implies that test edges $t1, t2, t3$ and $t4$, which start from a self-loop of v_1, cannot follow $t8$ in a realizable test sequence. The same restriction also applies to $t9, t10$, and $t11$.

The following test sequence is obtained by applying the rural Chinese postman method [16] to the graph without self-loop repetition constraint:

$$
\begin{aligned}
&\overbrace{e0, \mathbf{e1}, \mathbf{e5}}^{t0}, \overbrace{e12, \mathbf{e12}}^{t12}, \overbrace{e7, \mathbf{e13}}^{t7}, \overbrace{e13, \mathbf{e13}}^{t13}, \overbrace{e8, \mathbf{e0}, \mathbf{e2}}^{t8}, \overbrace{e1, \mathbf{e1}, \mathbf{e5}}^{t1}, e7, \overbrace{e9, \mathbf{e0}, \mathbf{e2}}^{t9}, \overbrace{e2, \mathbf{e1}, \mathbf{e5}}^{t2}, e7, \\
&\overbrace{e10, \mathbf{e0}, \mathbf{e2}}^{t10}, \overbrace{e3, \mathbf{e1}, \mathbf{e5}}^{t3}, e7, \overbrace{e11, \mathbf{e0}, \mathbf{e2}}^{t11}, \overbrace{e4, \mathbf{e1}, \mathbf{e5}}^{t4}, e7, e8, e0, \overbrace{e5, \mathbf{e12}}^{t5}, e7, e8, e0, \overbrace{e6, \mathbf{e13}}^{t6}, e8
\end{aligned}
\tag{27}
$$

The test sequence contains 47 edges (the edges that are part of UIO sequences appear in bold).

The following part of the above test sequence

Table 2
Test and ghost edges for the graph of Figure 7 (a)

Test edge	*Start vertex*	*End vertex*	*Edges included*
t0	v_0	v_2	e0, **e1, e5**
t1	v_1	v_2	e1, **e1, e5**
t2	v_1	v_2	e2, **e1, e5**
t3	v_1	v_2	e3, **e1, e5**
t4	v_1	v_2	e4, **e1, e5**
t5	v_1	v_2	e5, **e12**
t6	v_1	v_3	e6, **e13**
t7	v_2	v_3	e7, **e13**
t8	v_3	v_1	e8, **e0, e2**
t9	v_3	v_1	e9, **e0, e2**
t10	v_3	v_1	e10, **e0, e2**
t11	v_3	v_1	e11, **e0, e2**
t12	v_2	v_2	e12, **e12**
t13	v_3	v_3	e13, **e13**

$$..., \overbrace{e8, \mathbf{e0}, \mathbf{e2}}^{t8}, \overbrace{e1, \mathbf{e1}, \mathbf{e5}}^{t1}, e7, ... \tag{28}$$

requires that, after the IUT is brought into state v_1 via an edge $e0$, there should be enough time for at least three self-loop traversals before the IUT moves to another state. This part of the test sequence will fail after the second consecutive self-loop traversal. Since $max\_self(v_1) = 2$, the timeout edge $e6$ will be triggered instead of the required transition $e1$. The IUT will then move into v_3, thereby disrupting the test sequence.

To avoid disruption of the above test sequence due to timeouts, edge $t1$ must be prevented from following $t8$. To meet this requirement, the graph of Figure 7 (a) is converted by the method described in Section 6 to the graph shown in Figure 7 (b). The vertices for which conditions (18) and (19) hold (only v_1 in this example), are split and then connected by a single ghost edge. As can be seen in Figure 7 (b), test edges $t8, t9, t10$, and $t11$ may be followed only by edges $t5, e5, t6$, and $e6$. To test $t1, t2, t3$, and $t4$, vertex v_1 must be entered through a ghost edge $e0$.

By limiting the number of consecutive self-loop traversals in a state to the maximum allowable, the following test sequence for the graph of Figure 7 (b) is obtained:

$$\overbrace{e0, \mathbf{e1}, \mathbf{e5}}^{t0}, \overbrace{e12, \mathbf{e12}}^{t12}, \overbrace{e7, \mathbf{e13}}^{t7}, \overbrace{e13, \mathbf{e13}}^{t13}, \overbrace{e8, \mathbf{e0}, \mathbf{e2}}^{t8}, \overbrace{e6, \mathbf{e13}}^{t6}, \overbrace{e9, \mathbf{e0}, \mathbf{e2}}^{t9},$$

$$e6, \overbrace{e10, \mathbf{e0}, \mathbf{e2}}^{t10}, e6, \overbrace{e11, \mathbf{e0}, \mathbf{e2}}^{t11}, e6, e8, e0, \overbrace{e1, \mathbf{e1}, \mathbf{e5}}^{t1}, e7, e8, e0, \overbrace{e2, \mathbf{e1}, \mathbf{e5}}^{t2},$$
$$e7, e8, e0, \overbrace{e3, \mathbf{e1}, \mathbf{e5}}^{t3}, e7, e8, e0, \overbrace{e4, \mathbf{e1}, \mathbf{e5}}^{t4}, e7, e8, e0, \overbrace{e5, \mathbf{e12}}^{t5}, e7, e8 \tag{29}$$

The test sequence contains 56 edges, an increase of 9 edges or almost 20%.

The test sequence in Figure 7 (b) is minimum-length given the self-loop constraint, although it is longer than the absolute minimum-length test sequence in Figure 7 (a). The maximum allowed number of self-loop traversals is not exceeded in any visit to a vertex, ensuring that the test sequence is realizable in a test laboratory.

6.1 Application to other verification techniques

Several techniques have been proposed for use in the last step (i.e., the verification of an IUT's state) of an edge test, as defined in Section 3. In addition to the UIO sequences [18], the most well-known ones include the distinguishing sequences [19,20], and the characterizing (or W) sequences [6,19,20]. The results presented in this paper are based on using the UIO sequences as the state verification technique. However, these results are also applicable to distinguishing and characterizing sequences. Although a detailed solution for each technique is beyond the scope of this paper, the following highlights are provided to guide the reader on how to modify the solution given in this paper to include these state verification techniques.

Since a distinguishing sequence D also constitues a UIO sequence for each state of an FSM, the test edges in $G^{'}$ are created similar to those in the UIO sequences technique. The size of the augmented graph $G^{'}$ is the same for both techniques.

The UIO sequences are a special case of the characterizing sequences. In this paper, the UIO sequences are chosen to present the timing constraints problem, since using a form more general than the UIO sequences would only increase the notational complexity without providing any significant theoretical improvement.

In the characterizing sequences method, a characterizing set W contains a set of input sequences such that, when all sequences of $w_a \in W$ are applied to an IUT, each state is uniquely identified. The basic difference between modeling a state with characterizing sequences and the UIO sequences is that all $w_a \in W$ sequences must be considered for a state v_i when defining the test edges. As shown in Section 5.1, the UIO sequences technique creates one test edge for each edge $e = (v_i, v_j) \in E$ by using $UIO(v_j)$. In modeling with the characterizing sequences, however, there are $|W|$ test edges created in $G^{'}$ for each edge $e \in E$ (one test edge per $w_a \in W$). As a result, there are a total of $|W| * |E|$ test edges in $G^{'}$.

After constructing $G^{'}$ in this manner, the rest of the proposed solution can be directly applied. An example of applying the distinguishing and characterizing sequences to the rural Chinese postman problem formulation is presented in [33,34].

6.2 Fault coverage issues

A tradeoff exists between the length of test sequences and their fault coverage. The fault coverage of the test generation technique presented in this paper is expected to be the same as the fault coverage provided by the rural Chinese postman tours combined with the UIO sequences [16]. The fault coverage for the UIO sequences method is reported by Sidhu and Leung [35], and Sabnani and Dahbura [36]. They presented fault models based on the Monte Carlo simulation technique, where a given specification is randomly altered and checked by a test sequence. These studies concluded that test sequences generated by using the UIO sequences have a "high" fault detection capability.

In addition to the fault types studied in [35] and [36], this paper considers faults due to timers, as shown in the examples given in Section 6. In general, such faults due to disruption of a test sequence by unexpected timeouts may move an IUT into a wrong state (i.e., a state other than the one specified) or force an IUT to generate a wrong output to a given input (i.e., an output other than the one specified). Such events correspond to the errors where an IUT has incorrectly implemented a next state function or an output. The test steps shown in Section 3 combined with the UIO sequences are expected to detect such faults with the coverage estimated by [35,36].

7 Conclusion

An optimization method based on the Rural Chinese Postman Problem is introduced to generate test sequences with timing constraints. Due to the active timers, the number of consecutive self-loops that can be traversed in a given state before a timeout occurs is limited. A test sequence must consider this constraint to be realizable in a test laboratory.

In the solution presented here, a series of augmentations are defined for the directed graph representation of the deterministic FSM model of a protocol. The resulting test sequence is proven to be of minimum-length while not exceeding the tolerable limit of consecutive self-loops at each state. In addition to the UIO sequences method, the solution derived in this paper is also applicable to test sequences that use other state identification methods such as distinguishing sequences, and characterizing sequences.

Currently, this method is being implemented as a software tool to be applied to MIL-STD 188-220B [23].

References

[1] H. Ural, "Formal methods for test sequence generation," *Comput. Commun.*, vol. 15, pp. 311–325, June 1992.

[2] E. Brinksma, "A theory for the derivation of tests," in *Proc. IFIP Protocol Specif. Test. Verif. (PSTV)*, Amsterdam: North-Holland, 1988.

[3] J. Tretmans, "Conformance testing with labelled transitions systems: Implementation relations and test generation," *Comput. Networks ISDN Syst.*, vol. 29, no. 1, pp. 49–79, 1996.

[4] R. E. Miller and S. Paul, "On the generation of minimal-length conformance tests for communication protocols," *IEEE/ACM Trans. Networking*, vol. 2, pp. 116–129, Feb. 1993.

[5] R. E. Miller and S. Paul, "Structural analysis of protocol specifications and generation of maximal fault coverage conformance test sequences," *IEEE/ACM Trans. Networking*, vol. 2, pp. 457–470, Oct. 1994.

[6] G. Luo, G. von Bochmann, and A. F. Petrenko, "Test selection based on communicating nondeterministic finite state machines using a generalized Wp-method," *IEEE Trans. Softw. Eng.*, vol. 20, no. 2, pp. 149–162, 1994.

[7] B. Sarikaya, G. von Bochmann, and E. Cerny, "A test design methodology for protocol testing," *IEEE Trans. Softw. Eng.*, vol. 13, pp. 518–531, May 1987.

[8] R. J. Linn, "Conformance testing for OSI protocols," *Comput. Networks ISDN Syst.*, vol. 18, no. 3, pp. 203–219, 1990.

[9] B. Yang and H. Ural, "Protocol conformance test generation using multiple UIO sequences with overlapping," in *Proc. ACM SIGCOMM*, pp. 118–125, 1990.

[10] Y. N. Shen, F. Lombardi, and A. T. Dahbura, "Protocol conformance testing using multiple UIO sequences," *IEEE Trans. Commun.*, vol. 40, pp. 1282–1287, Aug. 1992.

[11] R. J. Linn and M. U. Uyar, *Conformance Testing Methodologies and Architectures for OSI Protocols.* Los Alamitos, CA: IEEE Comput. Soc. Press, 1994.

[12] W. Y. L. Chan and S. T. Vuong, "An improved protocol test generation procedure based on UIOs," in *Proc. ACM SIGCOMM*, Sept. 1989.

[13] H. Ural and Y. Lu, "An improved method for test sequence generation," Tech. Rep. TR-90-12, Dept. of CSI, Univ. of Ottawa, Mar. 1990.

[14] M. S. Chen, Y. Choi, and A. Kershenbaum, "Minimal length test sequences for protocol conformance," in *Proc. First Network Management and Control Workshop*, (New York, NY), 1989.

[15] M. S. Chen, Y. Choi, and A. Kershenbaum, "Approaches utilizing segment overlap to minimize test sequences," in *Proc. IFIP Protocol Specif. Test. Verif. (PSTV)*, pp. 85–98, 1990.

[16] 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. Commun.*, vol. 39, pp. 1604–1615, Nov. 1991.

[17] J. K. Lenstra and A. H. G. Rinnooy Kan, "On general routing problems," *Networks*, vol. 6, pp. 273–280, 1976.

[18] K. K. Sabnani and A. T. Dahbura, "A protocol test generation procedure," *Comput. Networks ISDN Syst.*, vol. 15, pp. 285–297, 1988.

[19] A. Bhattacharyya, *Checking Experiments in Sequential Machines.* New York, NY: Wiley & Sons, 1989.

[20] Z. Kohavi, *Switching and Finite Automata Theory.* New York, NY: McGraw-Hill, 1978.

[21] M. U. Uyar, M. A. Fecko, A. S. Sethi, and P. D. Amer, "Minimum-cost solutions for testing protocols with timers," in *Proc. IEEE Int'l Perf. Comput. Commun. Conf. (IPCCC)*, (Phoenix, AZ), pp. 346–354, Feb. 1998.

[22] AT&T 5E4 Generic Program, *AT&T* $5ESS^{TM}$ *Switch—ISDN Basic Rate Interface Specification*, Sept. 1985.

[23] DoD, *Military Standard—Interoperability Standard for Digital Message Device Subsystems (MIL-STD 188-220B)*, Jan. 1998.

[24] ISO, Information Technology—OSI, Geneva, Switzerland, *ISO International Standard 9646: Conformance Testing Methodology and Framework*, 1991.

[25] M. U. Uyar and M. H. Sherif, "Protocol modeling for conformance testing: Case study for the ISDN LAPD protocol," *AT&T Technical J.*, vol. 69, Jan. 1990.

[26] M. A. Fecko, P. D. Amer, A. S. Sethi, M. U. Uyar, T. Dzik, R. Menell, and M. McMahon, "Formal design and testing of MIL-STD 188-220A based on Estelle," in *Proc. IEEE Milit. Commun. Conf. (MILCOM)*, (Monterey, CA), Nov. 1997.

[27] L. R. Ford and D. R. Fulkerson, *Flows in Networks.* Princeton, NJ: Princeton University Press, 1962.

[28] M. U. Uyar and A. T. Dahbura, "Optimal test sequence generation for protocols: the Chinese postman algorithm applied to Q.931," in *Proc. IEEE GLOBECOM*, pp. 68–72, Dec. 1986.

[29] B. S. Bosik and M. U. Uyar, "FSM-based formal methods in protocol conformance testing: from theory to implementation," *Comput. Networks ISDN Syst.*, vol. 22, pp. 7–34, Sept. 1991.

[30] H. V. Bertine, W. B. Elsner, P. K. Verma, and K. T. Tewani, "Overview of protocol testing, methodologies, and standards," *AT&T Technical J.*, Jan. 1990.

[31] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[32] J. Springintveld, F. Vaandrager, and P. R. D'Argenio, "Testing timed automata," Tech. Rep. CTIT-97-17, Univ. of Twente, the Netherlands, 1997. (Invited talk at TAPSOFT'97, Lille, France, Apr 1997).

[33] H. Ural and K. Zhu, "Optimal length test sequence generation using distinguishing sequences," *IEEE/ACM Trans. Networking*, vol. 1, no. 3, pp. 358–371, 1993.

[34] A. Rezaki and H. Ural, "Construction of checking sequences based on characterization sets," *Comput. Commun.*, vol. 18, no. 12, pp. 911–920, 1995.

[35] D. P. Sidhu and T. K. Leung, "Fault coverage of protocol test methods," in *Proc. IEEE INFOCOM*, pp. 80–85, 1988.

[36] A. T. Dahbura and K. K. Sabnani, "An experience in estimating fault coverage of a protocol test," in *Proc. IEEE INFOCOM*, pp. 71–79, 1988.

Appendix

A Existence of polynomial-time solution

Below, we prove that if an FSM has a reset capability or a self-loop property, then $G[E^*_{test} \cup E^*_1]$ is a weakly-connected spanning subgraph of G^* (see Section 6 for definitions of E^*_{test} and E^*_1). In this case, any rural symmetric augmentation graph G^*_s of G^* is strongly connected and therefore has an Euler tour [16]. Note that a reset capability and a self-loop property are sufficient, but not necessary conditions for strong-connectivity of G^*_s.

Recall from Section 3 that state s_{init} is the initial state of an FSM representing a protocol. An FSM is said to have a *reset capability* if it can move into s_{init} from any other state with a single input operation. An FSM modeling a protocol is said to have a *self-loop property* if each vertex v_i of the graph G representing this FSM has at least one self-loop edge (v_i, v_i).

A.1 Reset capability

If an FSM has the reset property, then in a directed graph G representing this FSM there exist the initial vertex v_0 (corresponding to s_{init}) and an edge (v_i, v_0) for each vertex $v_i \neq v_0$.

Given the initial vertex v_0, let $\bar{v}^*_\theta$ be the ending vertex of $UIO(v_0)$. Now let us prove the following lemmas:

Lemma 1 *Each edge in $E^*_{test} \cup E^*_1$ incident on an arbitrary vertex $v_i^{*(2)}$ is weakly-connected with $\bar{v}^*_\theta$.*
Proof: From a reset capability of the protocol we obtain that there exists a test edge $(v_i^{*(2)}, \bar{v}^*_\theta) \in E^*_{test}$ that is a concatenation of the original reset edge $(v_i^{'}, v_0)$ and $UIO(v_0)$. Clearly, $\bar{v}^*_\theta$ is the root of a spanning tree including each vertex $v_i^{*(2)}$ (or v_i^*, if $v_i^{'}$ is not split). □

Lemma 2 *For each $v_i^{'}$ that is split by the algorithm, $v_i^{*(1)}$ and $v_i^{*(2)}$ are weakly-connected in $G[E^*_{test} \cup E^*_1]$.*
Proof: If $v_i^{'} \in V^I$, then $v_i^{*(1)}$ and $v_i^{*(2)}$ are connected via an edge in E^*_1. If $v_i^{'} \in V^{II} \cup V^{III}$, then $v_i^{'}$ is split only if the sets of outgoing test edges $E^s_{test}(v_i)$ and $E^{ns}_{test}(v_i)$ are not empty. Since any $(v_i^{*(1)}, \bar{v}^*_k) \in E^s_{test}(v_i)$ and any $(v_i^{*(2)}, \bar{v}^*_k) \in E^{ns}_{test}(v_i)$ end at the same vertex $\bar{v}^*_k$, $v_i^{*(1)}$ and $v_i^{*(2)}$ are weakly-connected via a pair of test edges in E^*_{test}. □

Lemma 3 *Each edge in $E^*_{test} \cup E^*_1$ incident on an arbitrary vertex $v_i^{*(1)}$ is weakly-connected with $\bar{v}^*_\theta$.*
Proof: From *Lemma* 2 we obtain that $v_i^{*(1)}$ is weakly-connected with $v_i^{*(2)}$. Then *Lemma* 1 implies that $v_i^{*(1)}$ and each edge incident on it are weakly-connected with $\bar{v}^*_\theta$. □

Lemma 4 *If an FSM has a reset capability, then $G[E^*_{test} \cup E^*_1]$ is a weakly-connected spanning*

subgraph of G^*.
Proof: Based on Lemmas 1 through 3 above, we conclude that $\bar{v}_\theta^*$ is the root of a spanning tree including each vertex $v_i^{*(1)}$ and $v_i^{*(2)}$ (or v_i^*, if vertex is not split). Therefore, $G[E_{test}^* \cup E_1^*]$ is a weakly-connected spanning subgraph of G^*. □

A.2 Self-loop property

Since the method presented in this paper is applied to graph G that consists of both test and ghost edges (Section 5.1), a self-loop property in this case means that each vertex has at least one *original* self-loop edge (which is equivalent to G having a self-loop ghost edge at each vertex).

Lemma 5 *It is not the case that there exists $v_i^{'}$ such that $v_i^{*(1)} \in V_a$ and $v_i^{*(2)} \in (V - V_a)$.*
Proof: Follows from the definition of V_a and *Lemma* 2. □

Lemma 6 *There is no $(\bar{v}_i^*, \bar{v}_j^*) \in E_{test}^*$ such that $\bar{v}_i^* \in V^a$ and $\bar{v}_j^* \in (V^* - V_a)$.*
Proof: Follows from the definition of V_a. □

Lemma 7 *There exists a ghost edge $(\bar{v}_i^*, \bar{v}_j^*)$ in G^* satisfying two conditions:*

1. *$\bar{v}_i^* \in V^a$ and $\bar{v}_j^* \in (V^* - V_a)$;*
2. *$(\bar{v}_i^*, \bar{v}_j^*)$ does not connect $v_i^{*(1)}$ with $v_i^{*(2)}$.*

Proof: The first condition follows immediately from strong-connectivity of G. The second condition follows from *Lemma* 5. □

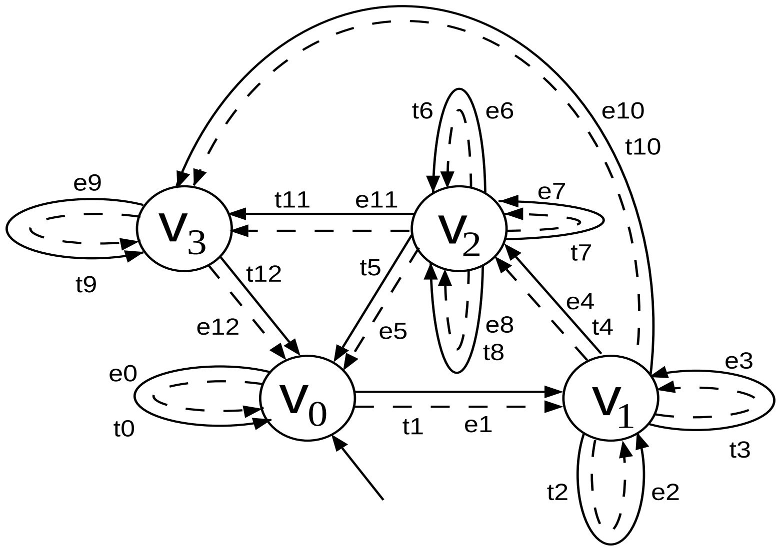
Lemma 8 *If an FSM has a self-loop property, then $G[E_{test}^* \cup E_1^*]$ is a weakly-connected spanning subgraph of G^*.*
Proof: BWOC, suppose that $G[E_{test}^* \cup E_1^*]$ is not a weakly-connected spanning subgraph of G^*. In this case, there exists a subset of vertices $V_a \subset V^*$ such that there is no path consisting of weakly-connected edges in $E_{test}^* \cup E_1^*$ that starts at $\bar{v}_i^* \in V^a$ and ends at $\bar{v}_j^* \in (V^* - V_a)$.

Let $(\bar{v}_i^*, \bar{v}_j^*)$ be a ghost edge satisfying conditions in *Lemma* 7. Then there exists a test edge $(\bar{v}_i^*, v_{uio}^j) \in E_{test}^*$ corresponding to $(\bar{v}_i^*, \bar{v}_j^*)$, where v_{uio}^j is the ending vertex of $UIO(v_j)$. *Lemma* 5 implies that in this case $\bar{v}_i^*, v_{uio}^j \in V_a$. Then

- **case (1)** $v_j^{'} \in V^{II} \cup V^{III}$**:** From the self-loop property and the definition of V_a, we obtain that there exists a test edge $(\bar{v}_j^*, v_{uio}^j)$ such that $\bar{v}_j^* \in (V^* - V_a)$ (*Lemma* 5) and $v_{uio}^j \in (V - V_a)$ (*Lemma* 6), which is a contradiction to $v_{uio}^j \in V_a$.
- **case (2)** $v_j^{'} \in V^I$**:** In this case G^* does not contain any self-loop test edges of $\bar{v}_j^*$. Since $UIO(v_j)$ is a self-loop, $v_{uio}^j = \bar{v}_j^*$. This implies that $(\bar{v}_i^*, \bar{v}_j^*) \in E_{test}^*$ is a test edge satisfying $\bar{v}_i^* \in V_a$ and $\bar{v}_j^* = v_{uio}^j \in (V^* - V_a)$, which is a contradiction to $v_{uio}^j \in V_a$.

We conclude that V_a defined above does not exist. Therefore, $G[E_{test}^* \cup E_1^*]$ is a weakly-connected spanning subgraph of G^*. □

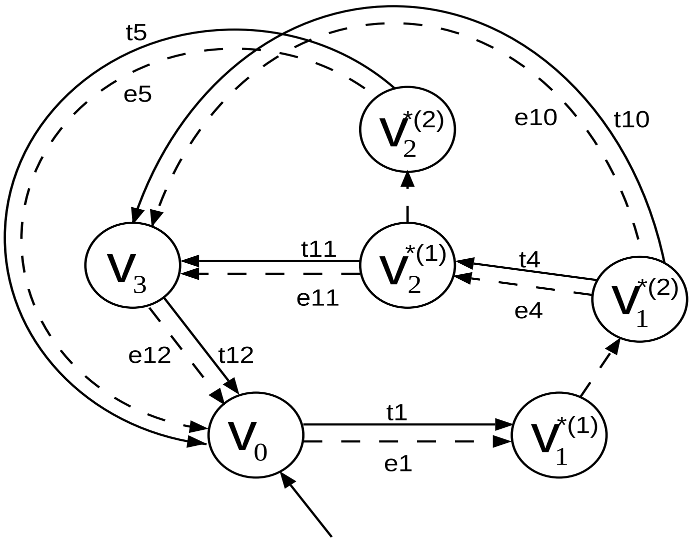
All UIO sequences are of Class 1.



(a)

Minimum-cost test sequence (34 edges)

e0 **e0** e1 **e2** e2 **e2** e10 **e9** e9 **e9** e12 **e0** e1 e3 **e2** e4 **e6**
e7 e6 **e6** **e7** e11 **e9** e12 e1 e4 e7 **e6** **e7** e8 **e6** **e7** e5 **e0**



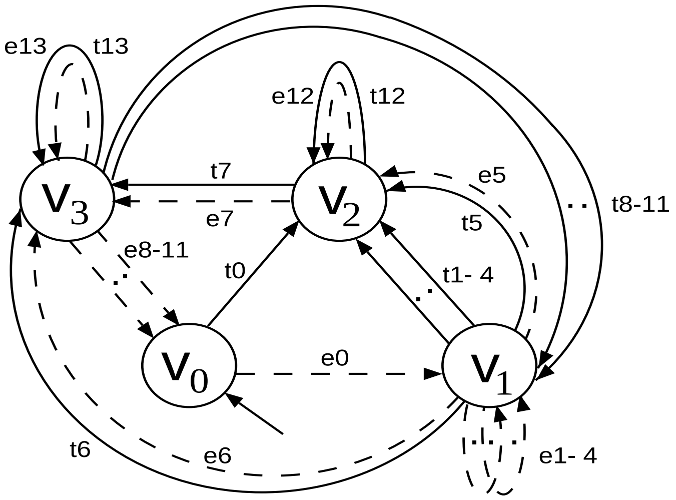
(b)

Minimum-cost test sequence (40 edges)

e0 **e0** e1 **e2** e10 **e9** e9 **e9** e12 **e0** e1 e2 **e2** e4 **e6** **e7** e11 **e9** e12
e1 e3 **e2** e4 e6 **e6** **e7** e5 **e0** e1 e4 e7 **e6** **e7** e5 e1 e4 e8 **e6** **e7** e5

Fig. 6. Minimum-cost test sequence (a) without and (b) with self-loop repetition constraint. Test and ghost edges appear in solid and dashed lines, respectively.

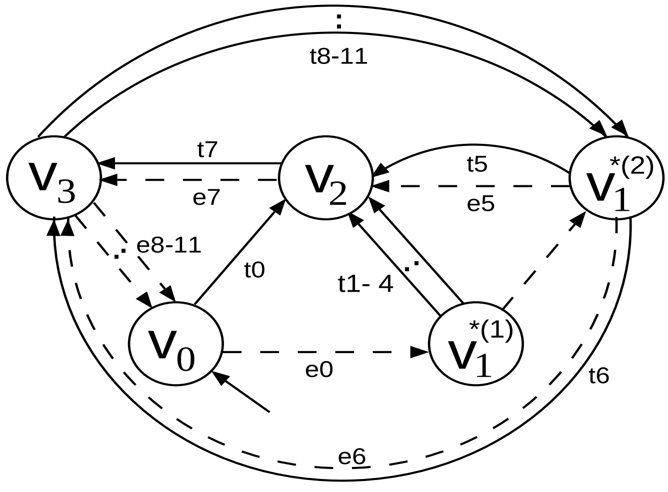
UIO sequences are of Classes 1, 2, and 3.



(a)

Minimum-cost test sequence (47 edges)

e0 **e1 e5** e12 **e12** e7 **e13** e13 **e13** e8 **e0 e2** e1 **e1 e5** e7 e9 **e0 e2** e2 **e1 e5** e7
e10 **e0 e2** e3 **e1 e5** e7 e11 **e0 e2** e4 **e1 e5** e7 e8 e0 e5 **e12** e7 e8 e0 e6 **e13** e8



(b)

Minimum-cost test sequence (56 edges)

e0 **e1 e5** e12 **e12** e7 **e13** e13 **e13** e8 **e0 e2** e6 **e13** e9 **e0 e2**
e6 e10 **e0 e2** e6 e11 **e0 e2** e6 e8 e0 e1 **e1 e5** e7 e8 e0 e2 **e1 e5**
e7 e8 e0 e3 **e1 e5** e7 e8 e0 e4 **e1 e5** e7 e8 e0 e5 **e12** e7 e8

Fig. 7. Minimum-cost test sequence (a) without and (b) with self-loop repetition constraint. Test and ghost edges appear in solid and dashed lines, respectively.