Minimum-Cost Solutions for Testing Protocols with Timers*

M. Ümit Uyar†
Department of Electrical Engineering
City College of New York, New York, NY

Mariusz A. Fecko, Adarshpal S. Sethi, Paul D. Amer
Computer and Information Science Department
University of Delaware, Newark, DE

Abstract

A method to generate a minimum-cost test sequence for a protocol with timers is presented. The protocol timers limit the number of consecutive self-loops that can be realized in a given state. The solution presented is applicable to test sequences that use any state identification method such as UIO sequences, distinguishing sequences, and characterizing sequences. If valid and inopportune transition testing are combined, or if only valid transitions are considered, a minimum-cost solution exists. In the case of testing inopportune transitions separately, however, finding a minimum-cost solution in shown to be NP-hard.

1 Introduction

As the complexity of communication protocols increases, testing implementations for conformance to their specifications has become an integral part of the product development cycle. Without the help of formal methods in protocol testing, the interoperability of devices is questionable. Various methods for automated test generation from protocol specifications have been proposed [1, 3, 5, 11, 12, 14, 17], based on a deterministic finite-state machine (FSM) model of the specification.

*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.

†Dr. Uyar, a Research Professor with CCNY, is presently Visiting Associate Professor at University of Delaware.

Although existing test generation methods concentrate on optimizing the test sequence length, these methods place no restrictions on the order in which the tests can be applied to an implementation under test (IUT). Possibly the most common restriction stems from an IUT's timers that, during actual testing, may limit the duration that the IUT can remain in a particular state. During testing, each state transition takes a certain time to realize. Any test sequence that traverses many consecutive self-loops easily can exceed the timer limits for the given state, and therefore not be realizable in a test laboratory (a *self-loop* represents a state transition that starts and ends at the same state).

Constraining the duration an IUT stays in a given state during testing is an important issue for test generation for large protocols which contain many self-loops in their FSM models. For example, protocols with status features such as ISDN Q.931 for supplementary voice services [22], or with many inopportune inputs such as MIL-STD 188-220A [24], have large numbers of self-loop transitions. In addition, this problem arises in protocols with test sequences that use unique input/output (UIO) sequences, distinguishing sequences [2, 9] or characterizing sequences [2, 9] that contain self-loops, such as LAPD [16].

This paper presents a solution to optimize test sequence length (and cost) under the constraint that an IUT can remain only for a limited amount of time in a given state during testing. Two different test suites are considered. The first suite combines the testing of valid and inopportune transitions [23]. An inopportune transition occurs when an IUT receives an input not expected in its current state. In general, an inopportune transition is modeled as a self-loop, the unexpected input is simply ignored. The second suite considers testing the valid and inopportune transitions separately.

This paper shows that there exists an optimal solution for the first suite and the valid transition testing part of the second suite. Optimizing the cost of a test sequence for the inopportune transition testing part of the second suite is NP-hard. A heuristic solution based on the Rural Postman Problem [10] is introduced.

This paper is organized as follows. Section 2 addresses the basics of test sequence generation and the practical restrictions due to the timers. Section 3 formulates the problem and a minimum-cost solution is presented in Section 4. Section 5 discusses issues related to optimal testing of inopportune transitions separately.

2 Preliminaries and practical restrictions on the test sequences

A *protocol* can be specified as a deterministic FSM [9, 15]:

$FSM =< S, I, O, \delta, \lambda, s_{init} >$ where

- $S = \{s_1, \ldots, s_n\}$ is a finite set of states
- $I = \{i_1, \ldots, i_n\}$ is a finite set of inputs
- $O = \{o_1, \ldots, o_n\}$ is a finite set of outputs
- δ : a state transition function that maps an input and current state to the next state: $\delta : S \times I \rightarrow S$
- λ : an output function that maps an input and current state to the output: $\lambda : S \times I \rightarrow O$
- s_{init} : the initial state of the FSM which is the state immediately after power-up.

State s_i is *equivalent* to state s_j if the inputs defined for s_i are a subset of those for s_j and the corresponding outputs are identical [13]. An FSM is said to be *minimal* if its specification does not have two or more equivalent states.

A directed graph $G = (V, E)$ can represent a deterministic FSM. 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 $L_k = a_l/o_m$ 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 can be 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. In this paper it is assumed that the costs of self-loops may be different, however, the time to traverse an edge is the same for all self-loops.

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. G is said to be *strongly-connected* if, for any pair of vertices v_i and v_j, v_j is reachable from v_i. Since a directed graph G representing a real-life protocol specification is strongly-connected, every state is reachable from the initial state, and from every state the initial state is reachable as well. If a graph is symmetric and strongly-connected, an Euler tour exists.

The so-called *Chinese Postman Problem* is defined as finding a minimum-cost tour of G that traverses every edge at least once [19]. The *Rural Postman Problem* is finding a minimum-cost tour for a subset of edges in G [10].

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 controlled and 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.

Several sequences have been proposed for use in the last step of the above single transition test, such as unique input-output (UIO) sequences (see Section 4.1), distinguishing sequences [2, 9] and characterizing (or W) sequences [2, 9]. A UIO sequence of a state v_i is a sequence of edges starting at v_i such that the output sequence generated by these edges is unique for v_i. Although this paper's results are applicable with all of these methods, the UIO sequences method is used for defining state verification sequences throughout the paper.

Aho et al. introduced an optimization for the test sequence length (and cost) using UIO sequences [1]. Shen et al. [15], Ural and Lu [18] 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. [6], Yang and Ural [21], and Miller and Paul [12].

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.

The problem of limiting the the number of consecutive self-loop traversals in a test sequence such as MIL-STD 188-220A [24] and LAPD [16], and for protocols with self-loop state verification sequences such as ISDN Q.931 for supplementary voice services (uses status feature which is a self-loop) [22] and LAPD (uses UIO sequences) [16].

In general, the majority of tests defined for an IUT are classified into two categories: valid and inopportune tests [11, 23]. 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 inopportune messages as self-loops with a null or warning output.

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. Two test suites are considered in this paper. In the first test suite, valid and inopportune tests are handled together. In the second test suite, the valid and inopportune tests are performed separately. Two test sequences are generated in this case - the first one tests valid transitions, whereas the second one only inopportune transitions.

A minimum-cost test sequence generation method for the first test suite is presented in Section 4. The test sequence generated by the presented algorithm is longer than an absolute minimum-cost test sequence that can be obtained without the self-loop restriction. The limitation on how long an IUT can stay in a state may force the IUT to visit a state several times more than otherwise necessary in an absolute minimum-cost tour.

The solution given in Section 4 also is applicable to the valid edge tests in the second test suite. However, to generate a minimum-cost test sequence for *inopportune edges only* is shown to be NP-hard in Section 5. Heuristics based on the Rural Postman Problem is introduced for this case.

3 Problem formulation

Let the directed 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 $v_i, v_j \in V$. Each edge $e \in E$ is labeled as either valid (e_{valid}) or inopportune ($e_{inoppportune}$).

Let us divide edges in E into three disjoint sets:

- *valid self-loop transitions:*
  $E_{vsl} = \{(v_i, v_j) : v_i, v_j \in V \ \wedge \ (v_i, v_j)_{valid} \in E \ \wedge \ v_i = v_j\}$
- *remaining valid transitions (valid non-self loops):*
  $E_{vnsl} = \{(v_i, v_j) : v_i, v_j \in V \ \wedge \ (v_i, v_j)_{valid} \in E \ \wedge \ v_i \neq v_j\}$
- *inopportune transitions:*
  $E_{inop} = \{(v_i, v_j) : v_i, v_j \in V \ \wedge \ (v_i, v_j)_{inopportune} \in E \ \wedge \ v_i = v_j\}$

As defined in Section 2, valid transitions are defined for the "normal" or expected behavior of the protocol entity, whereas inopportune transitions are defined for inputs arriving unexpectedly (or out of sequence) in a given state.

In general, besides the three sets being mutually disjoint, together they account for all transitions:

$$E = E_{vsl} \cup E_{vnsl} \cup E_{inop} \tag{1}$$

In cases where some inopportune messages are self-loops and the remaining ones are non-self-loop transitions (i.e., $E_{inop} = E_{inop\_self} \cup E_{inop\_non\_self}$), (1) becomes:

$$E = E_{vsl} \cup E_{vnsl} \cup E_{inop\_self} \cup E_{inop\_non\_self} \tag{2}$$

In this study, without loss of generality, we assume that (1) is valid. The results can easily be extended to the systems where (2) applies.

Let the out-degree and the in-degree of vertex v_i be defined respectively as:

$$\begin{aligned} d_{out}(v_i) &\stackrel{def}{=} card\{(v_i, v_j) : v_i, v_j \in V \ \wedge \ (v_i, v_j) \in E_{vnsl}\} \\ d_{in}(v_i) &\stackrel{def}{=} card\{(v_i, v_j) : v_i, v_j \in V \ \wedge \ (v_j, v_i) \in E_{vnsl}\} \end{aligned}$$

Note that by these definitions self-loops do not contribute to a vertex's degrees.

During testing, after traversing a given transition for the first time, state verification is performed. We first consider the special case where all UIO sequences are self-loops. The general case where UIO sequences may contain non-self-loops is presented in Section 4.1.

Let $d_{state\_ver}(v_i)$ be the number of self-loop transitions used to verify whether an IUT is in state v_i. Suppose that during testing, a given vertex $v_i \in V$ can tolerate at most $max\_self(v_i)$ self-loops executed at one visit to vertex v_i. As indicated in Section 2, attempting to remain in state v_i to execute $1 + max\_self(v_i)$ self-loops would result in disruption of a test sequence.

3.1 Constraints on the number of visits to a state

Let us consider a test suite where the valid and inopportune transition tests are combined and formally define the problem constraints. The case where the inopportune cases are tested separately is studied in Section 5.

Let E_{self} be the set of self-loop edges to be tested. Since valid and inopportune self-loops are tested together, $E_{self} \equiv E_{vsl} \cup E_{inop}$.

Let $d_{min\_self}(v_i)$ be the minimum number of times a tour covering all edges of $E_{vnsl} \cup E_{self}$ must include vertex $v_i \in V$. Let the number of self-loops of vertex v_i be defined as:

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

Intuitively, in a test sequence, at each visit to v_i, the maximum number of self-loops that can be traversed is $max\_self(v_i)$. Testing a self-loop transition involves traversing the self-loop transition followed by applying the state verification self-loop sequence, which contains $d_{state\_ver}(v_i)$ transitions ($d_{state\_ver}(v_i) > 1$ implies that the state verification self-loop sequence itself contains more than one self-loop transition).

Therefore, for each vertex v_i, the number of self-loop traversals is determined by two components. The first is the total number of self-loop traversals used for the purpose of state verification, which amounts to $d_{self}(v_i) * d_{state\_ver}(v_i)$. The second component is the number of self-loop transitions of a vertex v_i that are tested in a test sequence, i.e., $d_{self}(v_i)$.

Before any self-loop transition of a vertex v_i can be tested, an IUT must be brought to state v_i by traversing an edge (v_j, v_i). If it is the first occurrence of (v_j, v_i) in a tour, state verification for v_i is performed. To verify that the IUT is in state v_i, the state verification self-loop sequence is applied. This leaves $max\_self(v_i) - d_{state\_ver}(v_i)$ self-loop traversals that can be used for testing self-loop transitions of v_i at each visit to v_i after the first traversal of an edge (v_j, v_i) in the tour, and $max\_self(v_i)$ self-loop traversals at each subsequent visit to v_i through the edge (v_j, v_i).

To achieve minimum cost, we prefer a transition tour that does as much testing as possible when in a given state v_i without staying there too long. 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:

$$\Delta_1(v_i) \quad = \quad \lfloor \frac{max\_self(v_i) - d_{state\_ver}(v_i)}{1 + d_{state\_ver}(v_i)} \rfloor \tag{3}$$

Since there are exactly $d_{in}(v_i)$ non-self-loop edges with the ending state of v_i, the number of self-loop

transitions that can be tested during all visits to v_i after the first traversal of all non-self-loop edges (v_j, v_i) in the tour 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 $d_{self}(v_i)$, then if $d_{self}(v_i) \leq (d_{in}(v_i) * \Delta_1(v_i))$, all self-loop transitions can be tested during the required $d_{in}(v_i)$ visits to v_i. In this case

$$d_{min\_self}(v_i) \stackrel{def}{=} d_{in}(v_i) \tag{4}$$

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 visits to v_i through subsequent traversals of edges with the ending state of v_i is

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

During each visit through subsequent traversals of edges incoming to state v_i it is possible to test $\Delta_2(v_i)$ self-loops:

$$\Delta_2(v_i) \quad = \quad \lfloor \frac{max\_self(v_i)}{1 + d_{state\_ver}(v_i)} \rfloor \tag{5}$$

Note that (5) differs from (3) because no state verification of the transition entering state v_i is necessary.

Then $d_{min\_self}(v_i)$ is defined as

$$d_{min\_self}(v_i) \quad \stackrel{def}{=} \quad d_{in}(v_i) + \lceil \frac{d_{self}(v_i) - (d_{in}(v_i) * \Delta_1(v_i))}{\Delta_2(v_i)} \rceil \tag{6}$$

which applies when $d_{self}(v_i) > (d_{in}(v_i) * \Delta_1(v_i))$.

We conclude that the minimum number of times vertex v_i must be visited in a test sequence is defined by equation (7), which combines (4) and (6):

$$d_{min\_self}(v_i) \stackrel{def}{=} \begin{cases} d_{in}(v_i) & \text{if } d_{self}(v_i) \leq (d_{in}(v_i) * \Delta_1(v_i)) \\ d_{in}(v_i) + \lceil \frac{d_{self}(v_i) - (d_{in}(v_i) * \Delta_1(v_i))}{\Delta_2(v_i)} \rceil & \text{if } d_{self}(v_i) > (d_{in}(v_i) * \Delta_1(v_i)) \end{cases} \tag{7}$$

The upper bound of $max\_self(v_i)$ should allow testing at least one self-loop transition of v_i followed by the state verification self-loop sequence. This implies that (7) is valid only if

$$max\_self(v_i) \quad \geq \quad 1 + d_{state\_ver}(v_i) \tag{8}$$

If (8) is false, state verification cannot be applied. There is insufficient time to traverse a state verification sequence, and, in this case, step 3 of edge test procedure cannot be performed (or a shorter state

verification sequence must be utilized, if one exists). Even then, the method presented in this paper is valid. A constant of 0 must be substituted for $d_{state\_ver}(v_i)$ in (3) and (5). Equation (7) then simplifies to

$$d_{min\_self}(v_i) \stackrel{def}{=} \lceil \frac{d_{self}(v_i)}{max\_self(v_i)} \rceil \tag{9}$$

Let g be a function of two arguments: an edge $e \in E_{vnsl}$ 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 duplications of an edge $e \in E$ in G.

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

$$V^{'} = V \quad \wedge \quad E^{'} = E_{vnsl} \tag{10}$$

$$V^{''} = V \quad \wedge \quad E^{''} = E^{'} \cup E_g, \ where \ E_g \stackrel{def}{=} \bigcup_{e_{dup} \in E^{'}} g(e_{dup}, f(e_{dup})) \tag{11}$$

$$\forall v_i^{''} \in V^{''} d_{in}(v_i^{''}) = d_{out}(v_i^{''}) \tag{12}$$

$$\forall v_i^{''} \in V^{''} d_{in}(v_i^{''}) \geq d_{min\_self}(v_i) \tag{13}$$

The function f is the maximum-flow minimum-cost function defining the *symmetric augmentation* of $G^{'}$, which will be discussed in Section 4.

(11) and (12) define the symmetric graph $G^{''}$ as the symmetric augmentation of graph $G^{'}$. By definition, in $G^{''}$, the in-degree of any vertex $v_i^{''} \in V^{''}$ is equal to its out-degree. Also, equation (13) suggests that the in-degree of any vertex $v_i^{''} \in V^{''}$ be greater or equal to the value defined by $d_{min\_self}(v_i)$, where v_i is the corresponding vertex in V. †

Our goal is to build a Chinese Postman tour in which the constraint set by (13) is satisfied for each vertex $v_i^{'} \in V^{'}$. A Chinese Postman tour is a minimum-cost tour covering each transition $e \in E_{vnsl} \cup E_{self}$ at least once, and is equivalent to an Euler tour in a minimum cost symmetric graph $G^{''}$. In other words, our goal is to obtain the minimum-cost symmetric augmentation of the graph $G^{'}$ as the graph $G^{''}$. Therefore, this goal is now reduced to finding the value of the function f in equation (11) above for all edges in $E^{'}$.

†Note that, unless stated otherwise, $v_i^{'}, v_i^{''}, v_i^{*}, v_i^{\delta}$ and v_i^{σ} are used in this paper to denote the copies of a corresponding vertex $v_i \in V$ in graphs $G^{'}, G^{''}, G^{*}, G^{\delta}$ and G^{σ}, respectively.

4 Minimum-cost solutions for constrained self-loop testing

We present the following solution to the problem of finding a symmetric $G^{''}$ while satisfying the constraint set in (13). First, $G^{'}(V^{'}, E^{'})$ is converted to $G^{*}(V^{*}, E^{*})$ by splitting each vertex $v_i^{'} \in V^{'}$ satisfying

$$d_{min\_self}(v_i) \quad > \quad max(d_{in}(v_i), d_{out}(v_i)) \tag{14}$$

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

Then, $v_i^{*(1)}$ is connected to $v_i^{*(2)}$ with a set of edges with cardinality of $d_{min\_self}(v_i)$:

$$E_1^{*} \quad \stackrel{def}{=} \quad \bigcup_{v_i^{'} \in V^{'}} g((v_i^{*(1)}, v_i^{*(2)}), d_{min\_self}(v_i))$$

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

$$E_2^{*} \quad \stackrel{def}{=} \quad \{(v_i^{*(2)}, v_j^{*(1)}) : (v_i^{'}, v_j^{'}) \in E^{'}\}$$

The last step of the conversion is the addition of the source and sink vertices (s and t, respectively):

$$E^{*} \quad \stackrel{def}{=} \quad E_1^{*} \cup E_2^{*} \cup \bigcup_{v_i^{'} \in V^{'}} \{(s, v_i^{*(1)}), (s, v_i^{*(2)}), (v_i^{*(1)}, t), (v_i^{*(2)}, t)\}$$

The problem of finding the minimum-cost augmentation of $G^{'}$ as $G^{''}$ then can be reduced to finding the integer function $f : E \rightarrow N$ whose value $f(v_i^{*(2)}, v_j^{*(1)})$ determines the number of times an edge $(v_i^{'}, v_j^{'}) \in E^{'}$ needs to be duplicated to make the graph $G^{'}$ symmetric (Figure 1).

Aho et al. [1] presented an efficient solution to this problem for FSMs with either a self-loop property or a reset capability. We now apply a similar approach to the problem of minimizing the test sequence with the above self-loop repetition constraints.

We use network flow techniques to maximize the flow on graph G^{*} with minimum cost. Edges incident to the source and sink in G^{*} are assigned capacity β and a cost ψ as follows:

$$\beta(s, v_i^{*(1)}) \quad = \quad \max(0, d_{in}(v_i^{'}) - d_{min\_self}(v_i)) \tag{15}$$

$$\beta(s, v_i^{*(2)}) \quad = \quad \max(0, d_{min\_self}(v_i) - d_{out}(v_i^{'})) \tag{16}$$

$$\beta(v_i^{*(1)}, t) \quad = \quad \max(0, d_{min\_self}(v_i) - d_{in}(v_i^{'})) \tag{17}$$

$$\beta(v_i^{*(2)}, t) \quad = \quad \max(0, d_{out}(v_i^{'}) - d_{min\_self}(v_i)) \tag{18}$$

$$\psi(s, v_i^{*(1)}) \quad = \quad \psi(s, v_i^{*(2)}) = 0$$

$$\psi(v_i^{*(1)}, t) \quad = \quad \psi(v_i^{*(2)}, t) = 0$$

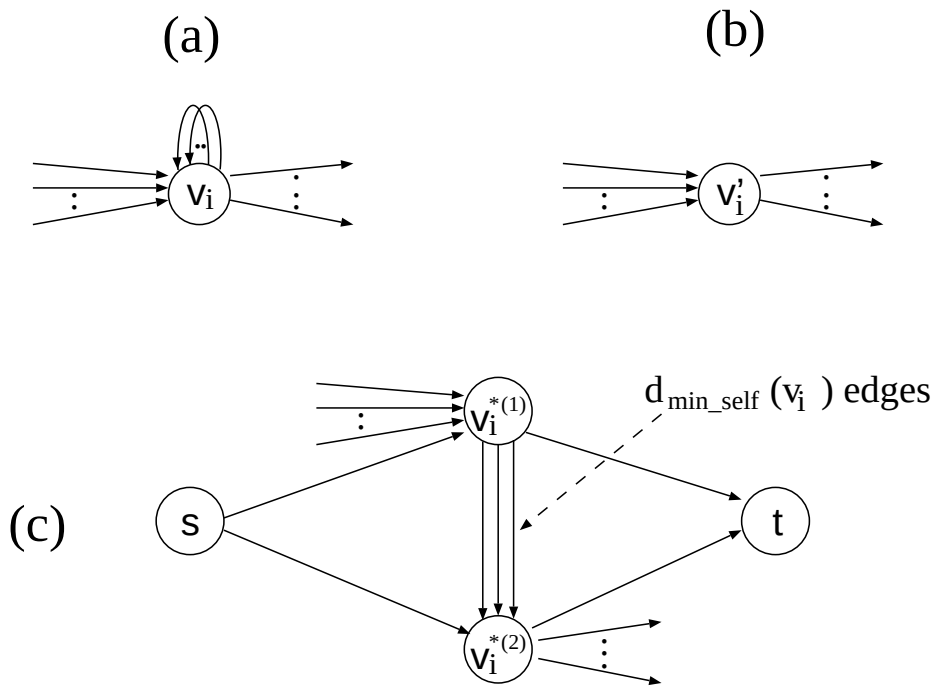


Figure 1: Conversion of v_i in G (part (a)), to $v_i^{'}$ in $G^{'}$ (part (b)) and to $v_i^{*(1)}, v_i^{*(2)}$ in G^* (part (c)).

Then, the capacity and cost assigned to the edges connecting split vertices are:

$$\forall (v_i^{*(1)}, v_i^{*(2)}) \in E_1^* \qquad \beta(v_i^{*(1)}, v_i^{*(2)}) = \infty$$
$$\forall (v_i^{*(1)}, v_i^{*(2)}) \in E_1^* \qquad \psi(v_i^{*(1)}, v_i^{*(2)}) = 0$$

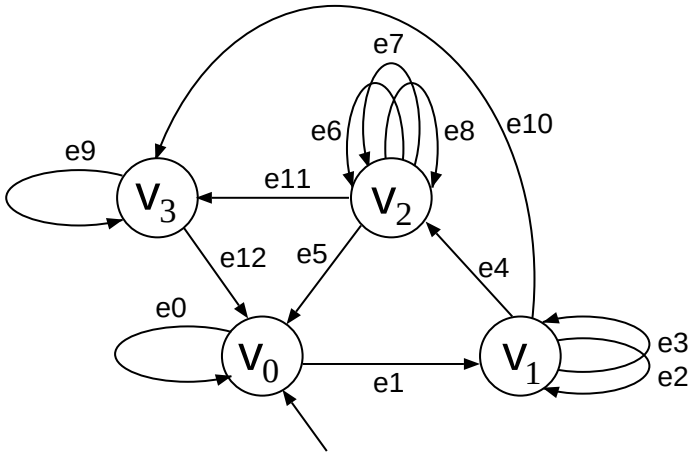
Each of the remaining edges in E^* (i.e., the edges corresponding to original edges in $E^{'}$) has infinite capacity with the cost of the original edge in $E^{'}$.

f is the maximum-flow minimum-cost function defined on the graph $G^*(V^*, E^*)$ that saturates all edges incident to s and t. Formally, such an f needs to satisfy the following conditions:

$$\begin{aligned}
(\forall v_i^* \in V^* - \{s,t\}) \sum_{v_j^* \in V^*} f(v_j^*, v_i^*) &= \sum_{v_j^* \in V^*} f(v_i^*, v_j^*) \\
(\forall v_i^* \in V^* - \{s,t\})\ \beta(s, v_i^*) &= f(s, v_i^*) \\
(\forall v_i^* \in V^* - \{s,t\})\ \beta(v_i^*, t) &= f(v_i^*, t)
\end{aligned}$$

The function f satisfying the above conditions exists iff

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



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**

Figure 2: Minimum-cost test sequence without self-loop repetition constraint.

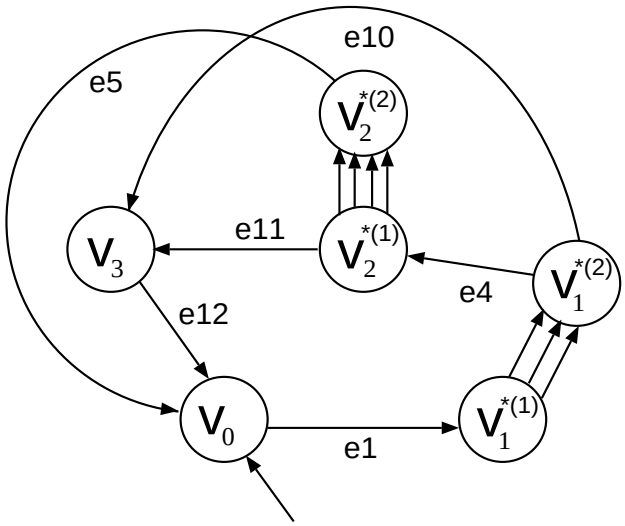
which holds true for capacity assignments defined by (15) through (18) [7].

It can be seen from Figure 1 that for each vertex split by the algorithm (i.e., each $v_i^{'}$ for which the condition (14) holds), a minimum of $d_{min\_self}(v_i)$ edges must be incident to the corresponding vertex $v_i^{''} \in V^{''}$ in a symmetric $G^{''}$.

Vertices that are not split by the algorithm (i.e., each $v_i^{'}$ for which the condition (14) does not hold) will have at least $max(d_{in}(v_i), d_{out}(v_i))$ incident edges after duplication, which is greater than or equal to $d_{min\_self}(v_i)$. Then in an Euler tour of $G^{''}$, each vertex $v_i^{''}$ will be visited at least $d_{min\_self}(v_i)$ times.

Example: Consider the FSM with self-loop transitions shown in Figure 2. Suppose that vertices v_0, v_2, and v_3 of the FSM can tolerate at most three, and 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. The outgoing edge $e12$ of v_3 corresponds to the timeout clear transition that resets the FSM back to the initial state v_0.

UIO sequences and the values of max_self, d_{state_ver} and d_{min_self} for vertices v_0, v_1, v_2, and v_3 are as follows:



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

Figure 3: Minimum-cost test sequence with self-loop repetition constraint.

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

The Chinese postman method [19] when applied to the graph without self-loop repetition constraint results in the test sequence

$$e0, \mathbf{e0}, e1, \mathbf{e2}, e2, \mathbf{e2}, e10, \mathbf{e9}, e9, \mathbf{e9}, e12, \mathbf{e0}, e1, e3, \mathbf{e2}, e4, \mathbf{e6},$$
$$\mathbf{e7}, e6, \mathbf{e6}, \mathbf{e7}, e11, \mathbf{e9}, e12, e1, e4, e7, \mathbf{e6}, \mathbf{e7}, e8, \mathbf{e6}, \mathbf{e7}, e5, \mathbf{e0} \quad (19)$$

containing 34 edges (the edges used for the purpose of 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 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 the test sequence will be disrupted.

Similarly, consider the following part of the test sequence (19):

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

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

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

where after three consecutive self-loops transitions e4,**e6**,**e7**, the sequence will prematurely take the IUT into state v_3. Again, the test sequence is disrupted by the timeout event.

To address the problem of test sequence disruption due to timeouts, the graph of Figure 2 is converted by the method described in Section 4 to the graph shown in Figure 3. The vertices for which condition (14) holds, which are v_1 and v_2, are split and then connected by $d_{min\_self}(v_1) = 3$ and $d_{min\_self}(v_2) = 4$ edges, respectively.

Considering the constrained self-loop problem, the test sequence for the graph of Figure 3 is obtained as

$$\begin{aligned} &e0, \mathbf{e0}, e1, \mathbf{e2}, e10, \mathbf{e9}, e9, \mathbf{e9}, e12, \mathbf{e0}, e1, e2, \mathbf{e2}, e4, \mathbf{e6}, \mathbf{e7}, e11, \mathbf{e9}, e12, \\ &e1, e3, \mathbf{e2}, e4, e6, \mathbf{e6}, \mathbf{e7}, e5, \mathbf{e0}, e1, e4, e7, \mathbf{e6}, \mathbf{e7}, e5, e1, e4, e8, \mathbf{e6}, \mathbf{e7}, e5 \end{aligned} \tag{20}$$

containing 40 edges.

Although the test sequence in Figure 3 is longer than that of Figure 2, it is minimum-length with the introduced 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 the test laboratory.

4.1 Results for non-self-loop UIO sequences

Recall from the discussion in Section 3.1 that, for a vertex $v_i \in V$, $d_{min\_self}(v_i)$ is the minimum number of times vertex v_i must be included in a tour covering all edges of $E_{vnsl} \cup E_{self}$. In the case of self-loop UIO sequences, $d_{min\_self}(v_i)$ is defined by equation (7). In the more general case, some UIO sequences may consist of both (or either) self-loop and non-self-loop edges.

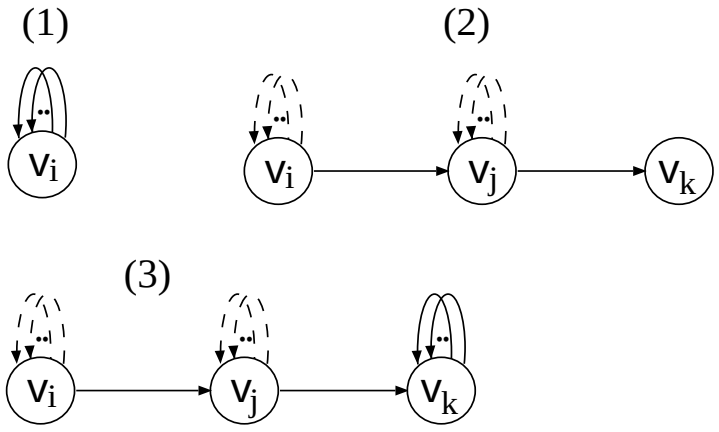


Figure 4: Three general classes of UIO sequences. The figure shows only edges that belong to $UIO(v_i)$.

A UIO sequence for vertex v_i (which we denote as $UIO(v_i)$) can be viewed as a concatenation of a number of subsequences:

$$UIO(v_i) = uio(v_i, v_i) \cdot uio(v_i, v_{j_0}) \cdot uio(v_{j_0}, v_{j_0}) \cdot ... \cdot uio(v_{j_{m-1}}, v_{j_m}) \cdot uio(v_{j_m}, v_{j_m}) \tag{21}$$

where $\cdot$ is a concatenation operator. In (21), $uio(v_{j_k}, v_{j_k})$ denotes the subsequence that contains only self-loop edges of v_{j_k}. A subsequence of $uio(v_{j_{k-1}}, v_{j_k})$ is a path of non-self-loop edges starting at $v_{j_{k-1}}$ and ending at v_{j_k}.

Based on this definition, in general, there are three possible forms that a UIO sequence can have. For each of these general forms, the applicability of the results is briefly discussed below.

Class 1. $UIO(v_i) = uio(v_i, v_i)$

The $UIO(v_i)$ is a self-loop UIO sequence, and $d_{min\_self}(v_i)$ is defined by equation (7). An example is given in Figure 4 (1). This is the same case that is discussed in Sections 3 and 4. If all UIO sequences belong to Class 1, after obtaining graph G^*, the Chinese Postman tour can be found in polynomial-time as described in Section 4.

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

In this class, the $UIO(v_i)$ may or may not start with a self-loop edge sequence, but it ends with a non-self-loop edge (see Figure 4 (2)). In this case, every time a self-loop edge is tested, the UIO sequence moves the IUT out of v_i. Therefore, v_i must be visited at least $d_{min\_self}(v_i)$ times:

$$d_{min\_self}(v_i) \stackrel{def}{=} d_{in}(v_i) + d_{self}(v_i)$$

This class requires that vertex v_i satisfy $max\_self(v_i) \geq 1 + |uio(v_i, v_i)|$. Moreover, each vertex v_{j_k} such that $uio(v_{j_k}, v_{j_k}) \subset UIO(v_i)$ must satisfy $max\_self(v_{j_k}) \geq |uio(v_{j_k}, v_{j_k})|$. If UIO sequences of G belong to either Class 1 or Class 2, after obtaining G^*, a minimum-cost test sequence can be calculated as described in [1].

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

In 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 also may contain self-loop edges in the middle. An example is in Figure 4 (3). The calculation of $d_{min\_self}(v_{j_k})$ requires that all UIO sequences ending at v_{j_k} must be known. The value of $d_{min\_self}(v_{j_k})$ also depends on the class of $UIO(v_{j_k})$. The calculation of $d_{min\_self}(v_{j_k})$ for Class 3 is more complicated, which is currently being studied [20]. Once $d_{min\_self}(v_{j_k})$ is calculated, the value of d_{min_self} can be obtained for all UIO sequences. Afterwards, graph G^* can be obtained from G and the test sequence can be generated as described in [20].

5 Testing of inopportune transitions separately

We now consider the case where the valid and inopportune transitions are to be tested separately. The solution presented in the previous section is directly applicable to the valid transitions of this test suite. However, minimum-cost test generation for only inopportune transitions needs revisiting. Since, separated from the valid transitions, inopportune transitions alone cover only a subset of the edges, finding the minimum-cost solution becomes more difficult.

Let the set of inopportune self-loops of vertex $v_i \in V$ be defined as:

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

During testing, we build a tour that contains all inopportune self-loop transitions, which are distributed over a subset of vertices of G. To be able to bring an IUT into a state with self-loop transitions, we must include a subset of valid non-self-loop transitions in a transition tour. Each time a vertex v_i is visited, at most $max\_self(v_i)$ self-loop edges are traversed. Testing of each inopportune self-loop transition is followed by executing the state verification self-loop sequence, which contains $d_{state\_ver}(v_i)$ transitions. Therefore, the number of inopportune transitions that can be tested during each visit to v_i is defined as:

$$\Delta_3(v_i) \quad = \quad \lfloor \frac{max\_self(v_i)}{1 + d_{state\_ver}(v_i)} \rfloor \tag{22}$$

Then the test sequence must visit v_i at least $d_{min\_inop}(v_i)$ times:

$$d_{min\_inop}(v_i) \stackrel{def}{=} \lceil \frac{|E_{inop}(v_i)|}{\Delta_3(v_i)} \rceil \tag{23}$$

Let $V_{self} \subseteq V$ be the set of vertices that have at least one inopportune self-loop transition:

$$V_{self} \stackrel{def}{=} \{v_i : v_i \in V \ \wedge \ d_{min\_inop}(v_i) > 0\}, V_{self} \subseteq V \tag{24}$$

The problem of testing inopportune transitions in an optimal way is now reduced to finding a minimum cost transition tour in $G^{'}(V^{'}, E^{'})$ that consists of a subset of E_{vnsl} and visits each vertex in $v_i \in V_{self}$ at least $d_{min\_inop}(v_i) > 0$ times.

5.1 NP-hardness of the problem

We first prove that, for a given graph $G^{'}(V^{'}, E^{'})$, the problem of finding a minimum-cost tour that includes each vertex $v_i^{'} \in V_{self} \subseteq V^{'}$ at least $d_{min\_inop}(v_i)$ times, posed in Section 5 (referred to as P_{inop} henceforth) is NP-hard. We will show that the Rural Postman Problem (RPP), which is known to be NP-complete for the most general case [10], is polynomial-time reducible to P_{inop}.

Recall from Section 2 that, for a given directed graph $G(V, E)$ and a subset of edges $E_c \subseteq E$, a Rural Postman Tour is a tour such that each edge in E_c is traversed at least once. The Rural Postman Problem is to find a rural postman tour of a minimum cost.

The reduction algorithm begins with an instance of RPP. Let $G(V, E)$ be an arbitrary directed graph. Let E_c be the subset of edges $E_c \subseteq E$ that a solution to RPP is supposed to include at least once, and let the solution to RPP have the cost with an upper bound of Ψ.

Now we build an instance of P_{inop}. Let $\widehat{G}$ $(\widehat{V}, \widehat{E})$ be the graph derived from G by replacing each edge $(v_i, v_j) \in E_c$ with two edges (v_i, u_{ij}) and (u_{ij}, v_j) such that $\psi(v_i, u_{ij}) = \psi(v_i, v_j)$ and $\psi(u_{ij}, v_j) = 0$.

Graph $\widehat{G}$ can easily be obtained in polynomial time. As an example of this construction, consider the directed graph in Figure 5. The edge $(v_1, v_2) \in E_c$ is converted to a vertex u_{12} and two edges $(v_1, u_{12}), (u_{12}, v_2) \in \widehat{E}$.

Thus we obtained the instance of P_{inop}, in which each vertex u_{ij} introduced in the conversion from G to $\widehat{G}$ has to be visited at least once (i.e., $d_{min\_inop}(u_{ij}) = 1$) in a tour of the cost upper-bounded by Ψ. Let U be the set of all vertices u_{ij}.

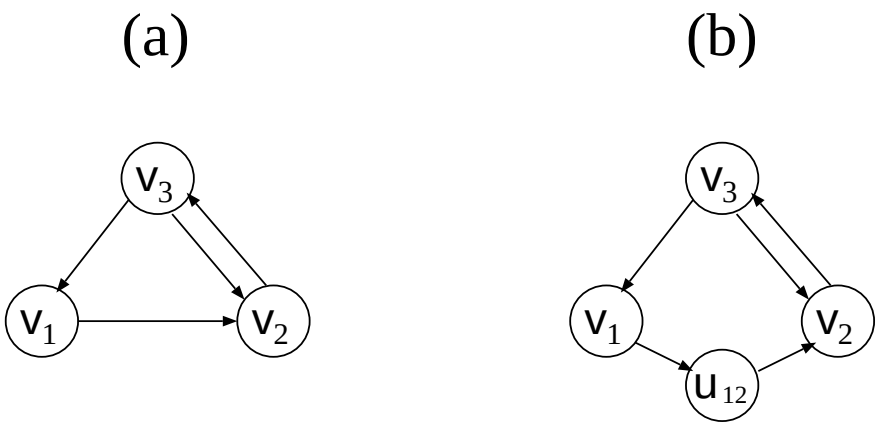


Figure 5: Conversion from an edge $(v_1, v_2) \in E_c$ in G (part (a)) to edges (v_1, u_{12}) and (u_{12}, v_2) in $\widehat{G}$ (part (b)).

We must show that this transformation of RPP into P_{inop} is a reduction [4]. First, suppose that T as a solution to RPP is a transition tour of G of the cost upper-bounded by Ψ, and that T covers each $(v_i, v_j) \in E_c$ at least once. Then it is clear that by replacing each $(v_i, v_j) \in E_c$ in T with the corresponding edges $(v_i, u_{ij}), (u_{ij}, v_j)$, we obtain the tour $\widehat{T}$ that visits each vertex $u_{ij} \in U$ at least once, and whose cost $\psi(\widehat{T}) = \psi(T) \leq \Psi$.

Conversely, suppose that $\widehat{T}$ is a solution to P_{inop}. By definition, $\widehat{T}$ contains each vertex u_{ij} at least once, and therefore contains all tuples (v_i, u_{ij}, v_j) at least once. By removing u_{ij} from each tuple (v_i, u_{ij}, v_j), we obtain a solution to RPP. Since $\psi(\widehat{T}) \leq \Psi$, and $\psi(T) = \psi(\widehat{T})$, then T is a tour of G with the cost less than or equal to Ψ.

Therefore, based on the above discussion, it can be concluded that P_{inop} is NP-hard.

5.2 Heuristics for inopportune self-loops problem based on RPP

Let us consider a heuristic solution to P_{inop} based on the Rural Postman Problem. First, the graph $G'(V', E')$, defined in Section 3.1 by equation (10), is converted to a $G^{\delta}(V^{\delta}, E^{\delta})$ by splitting each vertex $v_i' \in V_{self} \subseteq V'$, where V_{self} is defined in (24), into the two vertices $v_i^{\delta(1)}$, $v_i^{\delta(2)} \in V^{\delta}$. Then, $v_i^{\delta(1)}$ is connected to $v_i^{\delta(2)}$ with a set of $d_{min\_inop}(v_i)$ edges of infinite capacity and zero cost (Figure 6).

The size of G^{δ} is as follows:

$$|V^{\delta}| \quad = \quad |V'| + |V_{self}|$$

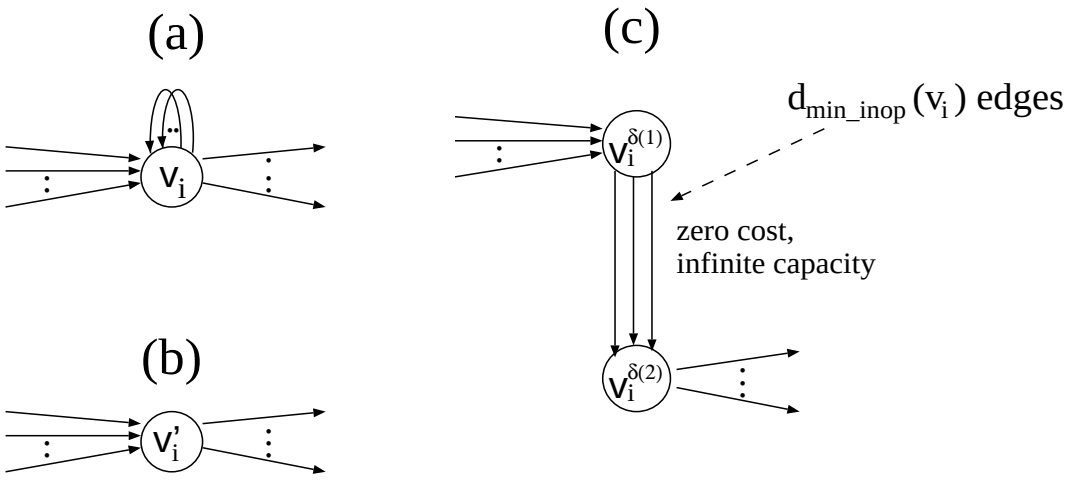


Figure 6: Conversion of $v_i \in V_{self}$ in G (part (a)), to $v_i^{'}$ in $G^{'}$ (part (b)), and to $v_i^{\delta(1)}, v_i^{\delta(2)}$ in G^{δ} (part (c)).

$$|E^{\delta}| \quad = \quad |E^{'}| + \sum_{v_i \in V_{self}} d_{min\_inop}(v_i)$$

Let E_c^{δ} be the set of new edges added to the graph. A heuristic algorithm to solve the RPP defined on G^{δ} and the subset $E_c^{\delta} \subseteq E^{\delta}$ gives a minimum-cost transition tour T^{δ}, which includes each $(v_i^{\delta(1)}, v_i^{\delta(2)}) \in E_c^{\delta}$ at least once. Since there are exactly $d_{min\_inop}(v_i)$ edges in E_c^{δ} corresponding to each $v_i^{'} \in V_{self}$, then by replacing each tuple $((v_p^{\delta}, v_i^{\delta(1)})\ (v_i^{\delta(1)}, v_i^{\delta(2)})\ (v_i^{\delta(2)}, v_q^{\delta}))$ in T^{δ} with $((v_p^{\delta}, v_i^{'})\ (v_i^{'}, v_q^{\delta}))$, we obtain a heuristic solution to P_{inop}.

For a class of graphs where E_c forms an edge-induced, weakly-connected subgraph of E^{δ}, a low-degree polynomial-time solution exists [8, 10]. However, it can be shown that E_c^{δ} does not satisfy this condition. Therefore, P_{inop} has only a suboptimal heuristic solution.

6 Conclusion

This paper describes a method to generate a minimum-cost test sequence with the constraint that only a limited number of consecutive self-loops can be realized in a given state. This constraint for example may be active timers that are running in certain states. The final test sequence is longer than the absolute minimum length due to the additional constraints imposed on the optimization problem by the timers. The test length can be further shortened by the segment overlapping techniques of Chen et al. [6], Yang and Ural [21], and Miller and Paul [12].

The solution presented in this paper is applicable to test sequences that use various state identification methods such as UIO sequences, distinguishing sequences, and characterizing sequences.

If the valid and inopportune transition testing are combined, or only the valid transitions are considered, a minimum-cost solution exists. The minimum-cost test sequence generation for testing the inopportune transitions separately is shown to be NP-hard.

As future work, this method will be implemented as a software tool and be applied to MIL-STD 188-220A [24].

References

[1] A.V. Aho, A.T. Dahbura, D. Lee, M.U. Uyar. *An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours.* IEEE Transactions on Communications, Vol. 39, No. 11, Nov.1991.

[2] A. Bhattacharyya. *Checking Experiments in Sequential Machines.* John Wiley & Sons, New York, N.Y., 1989.

[3] B. S. Bosik and M. U. Uyar. *FSM-Based Formal Methods in Protocol Conformance Testing: from Theory to Implementation.* Computer Networks and ISDN Systems, Vol. 22, No.1, 7-33, Sept 1991.

[4] T.H. Cormen, C.E. Leiserson, E.L. Rivest. *Introduction to Algorithms.* McGraw-Hill, New York, 1992.

[5] W.Y.L. Chan, S.T. Vuong. *An Improved Protocol Test Generation Procedure Based on UIOs.* Proc. ACM SIGCOM, Sept 1989.

[6] M. S. Chen, Y. Choi, A. Kershenbaum. *Minimal Length Test Sequences for Protocol Conformance.* Proc. First Network Management and Control Workshop, Polytechnic University, New York, NY, 1989.

[7] L.R. Ford, D.R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.

[8] A. Gibbons. *Algorithmic Graph Theory.* Cambridge, MA: Cambridge University Press, 1985.

[9] Z. Kohavi. *Switching and Finite Automata Theory.* McGraw Hill, New York, N.Y., 1978.

[10] J.K. Lenstra, A.H.G. Rinnooy Kan. *On General Routing Problems.* Networks, Vol. 6, 273-280, 1976.

[11] R.J. Linn. *Conformance Testing for OSI Protocols.* Computer Networks and ISDN Systems, No. 18, 1989/90, pp. 203-219.

[12] R. Miller, S. Paul. *Generating Maximal Fault Coverage Conformance Test Sequences of Reduced Length for Communication Protocols.* Proc. Int. Conference on Network Protocols, San Francisco, CA, Oct 1993.

[13] K.K. Sabnani, A.T. Dahbura. *A Protocol Test Generation Procedure.* Computer Networks and ISDN Systems, Vol. 15, 1988, 285-297.

[14] B. Sarikaya, G. Bochmann, E. Cerny. *A Test Design Methodology for Protocol Testing.* IEEE Trans. Software Engineering, Vol. SE-13, No. 5, May 1987, pp. 518-531.

[15] Y.N. Shen, F. Lombardi, A.T. Dahbura. *Protocol Conformance Testing Using Multiple UIO Sequences.* IEEE Trans. on Communications, vol. 40, no 8, pp. 1282-1287, Aug. 1992.

[16] M.H. Sherif, M.U. Uyar. *Protocol Modeling for Conformance Testing: Case Study for the ISDN Protocol.* AT&T Technical Journal, Jan/Feb 1990.

[17] H. Ural. *Formal Methods for Test Sequence Generation.* Computer Communications, vol 15, no. 5, pp. 311-325, June 1992.

[18] H. Ural, Y. Lu. *An Improved Method for Test Sequence Generation.* Technical Report, TR-90-12, Dept. of CSI, University of Ottawa, March 1990.

[19] M.U. Uyar, A.T. Dahbura. *Optimal Test Sequence Generation for Protocols: The Chinese Postman Algorithm Applied to Q.931.* Proc. IEEE Global Comm. Conf., 1986, 68-72.

[20] M.U. Uyar, M.A. Fecko, A.S. Sethi, P.D. Amer. *Test Generation for Protocols with Timing Constraints.* In preparation.

[21] B. Yang, H. Ural. *Protocol Conformance Test Generation Using Mulitple UIO Sequences with Overlapping.* Proc. ACM SIGCOMM'90, pp.118-125, 1990.

[22] *AT&T* $5ESS^{TM}$ *Switch - ISDN Basic Rate Interface Specification.* 5E4 Generic Program, AT&T 5D5-900-301, Sep. 1985.

[23] *Information Processing Systems - OSI Conformance Test Methodology and Framework.* ISO/IEC JTC 1, IS 9646, 1991.

[24] *Military Standard - Interoperability Standard for Digital Message Device Subsystems (MIL-STD 188-220A), 27Jul96.*