Hardness of deriving invertible sequences from finite state machines

Robert M. Hierons[1], Mohammad Reza Mousavi[2], Michael Kirkedal Thomsen[3], and Uraz Cengiz Türker[4]

[1] Department of Computer Science, Brunel University London, United Kingdom. rob.hierons@brunel.ac.uk

[2] Center for Research on Embedded Systems (CERES), School of IT, Halmstad University, Halmstad, Sweden m.r.mousavi@hh.se

[3] Department of Computer Science, University of Copenhagen, Denmark m.kirkedal@di.ku.dk

[4] Computer Engineering, Faculty of Engineering, Gebze Technical University, Kocaeli, Turkey urazc@gtu.edu.tr

Abstract. Many test generation algorithms use unique input/output sequences (UIOs) that identify states of the finite state machine specification M. However, it is known that UIO checking the existence of UIO sequences is PSPACE-complete. As a result, some UIO generation algorithms utilise what are called invertible sequences; these allow one to construct additional UIOs once a UIO has been found. We consider three optimisation problems associated with invertible sequences: deciding whether there is a (proper) invertible sequence of length at least K; deciding whether there is a set of invertible sequences for state set S' that contains at most K input sequences; and deciding whether there is a single input sequence that defines invertible sequences that take state set S'' to state set S'. We prove that the first two problems are NP-complete and the third is PSPACE-complete. These results imply that we should investigate heuristics for these problems.

1 Introduction

Software testing is an indispensable yet costly part of the development lifecycle and this has led to interest in test automation. Model based testing (MBT) is a high-profile approach to automation. It assumes the presence of a model that represents the abstraction of some aspect of the expected behaviour of the *system under test (SUT)*. The model is usually represented as an extended finite state machine, a finite state machine, or a labelled transition system.

In MBT, it is normal to generate test cases from a given model/specification M. A test case is then applied to M and the response (the expected behaviour) of M is recorded. The test case is then executed on the SUT N and the response (observed behaviour) is recorded. If the expected behaviour and observed behaviour differ then the tester declares that the SUT failed the test. Otherwise, the tester declares that the SUT passed the test case.

A number of techniques have been developed for generating test cases from an FSM, with this line of research dating back to the seminal papers of Moore [1] and Hennie [2]. Although FSM-based test generation techniques vary, they typically aim to *test transitions*, where a transition is a tuple (s, x, y, s') specifying that if M receives input x when in state s then it moves to s' and outputs y. In order to test a transition τ of SUT N, it is necessary to bring N to a state from which τ can be executed, fire the transition, record its output and decide whether the resultant state of the SUT is the expected state. Most such techniques use *state identification sequences* for the last part of this procedure [2–8]. The most widely used state identification sequences are *distinguishing sequences* (DSs) [9], *unique input output sequences* (UIOs) [10] and *characterising sets* (CSs) [10].

There are two types of DSs. A *Preset Distinguishing Sequence (PDS)* and an *Adaptive Distinguishing Sequence (ADS)* (also known as a *Distinguishing Set* [11]). When applied, DSs lead to different output sequences from the different states of M. One important property of DSs is that it has been known that it is possible to construct test sequences in polynomial time [12].

However, it has been long known that an FSM need not have a DS and instead one might use a UIO for a state s': an input sequence that distinguishes s' from all other states of M but need not distinguish any other pairs of states of M. Although not all FSMs have a UIO for every state, it has been reported that in practice most FSMs do have such UIOs [3] and this has led to the development of many FSM-based test generation methods that use UIOs [3, 13–20]. However, the problem of checking the existence of a UIO is PSPACE-hard [21].

A CS is a set of input sequences that distinguish all pairs of states and it has been shown that every minimum FSM has a CS [22, 4]. Another appealing aspect of CSs is that one can compute a CS from a given FSM in polynomial time [22, 4, 23]. However, experiments suggest that the use of CSs can lead to relatively long tests [12].

1.1 Motivation and Problem Statement

When generating test cases from an FSM it is desirable to have techniques that reduce the time spent on deriving state identification sequences and there has thus been work on this problem [24, 6, 25, 12, 26]. One promising method is to use *invertible sequences*[5] [27, 28]. Despite this, to our knowledge there is no work that investigates the problem of computing invertible sequences.

In this paper, we first extend the notion of invertibility to sets of states. Then we introduce optimisation problems related to invertible sequences, with these being motivated by a desire to reduce the cost of generating state identification sequences. Finally, we determine the computational complexity of these problems.

[5] An invertible sequence is a walk ρ with the property that if one determines the ending state of ρ then one also determines the starting state of ρ. In the following sections we formally define invertible sequences.

1.2 Structure of the paper

This paper is organised as follows. Section 2 defines FSMs and the corresponding notation, while Section 3 defines invertible sequences and the decision problems in which we are interested. In Section 4, we derive the bounds for the three decision problems considered. In Section 5, we draw conclusions and discuss possible lines of future work.

2 Preliminaries

In this section, we introduce some terminology related to finite state machines.

Definition 1. *A deterministic FSM is defined by a tuple* $M = (S, s_0, X, Y, \delta, \lambda)$*, where:* $S = \{s_1, s_2, \ldots, s_n\}$ *is the finite set of states;* $s_0 \in S$ *is the initial state;* $X = \{x_1, x_2, \ldots, x_r\}$ *is the finite set of inputs;* $Y = \{y_1, y_2, \ldots, y_v\}$ *is the finite set of outputs (*X *is disjoint from* Y*);* $\delta : S \times X \to S$ *is the transition function; and* $\lambda : S \times X \to Y$ *is the output function.*

Throughout this paper, $M = (S, s_0, X, Y, \delta, \lambda)$ denotes an FSM from which test sequences are to be generated. At any given time, M is in a state from S and accepts one input at a time. If an input $x \in X$ is applied when M is in state s then M changes its state to $\delta(s, x)$ and produces output $\lambda(s, x)$. We say that $\tau = (s, x, y, s')$ is a *transition* of M with *starting state* s, *ending state* s', and *label* x/y. The label x/y has *input portion* $(in(x/y))$ x and *output portion* y.

Given sequences $\bar{x}$ and $\bar{x}'$, $\bar{x}\bar{x}'$ denotes the concatenation of $\bar{x}$ and $\bar{x}'$. We use $pre(.)$ $(post(.))$ to denote the set of prefixes (suffixes). Given input/output pairs $x_1/y_1, \ldots, x_k/y_k$ we use $x_1/y_1 \ldots x_k/y_k$ and also $x_1x_2 \ldots x_k/y_1y_2 \ldots y_k$ to denote the corresponding input/output sequence. Further, we let $x_1 \ldots x_k$ and $y_1 \ldots y_k$ denote the *input portion* $(in(x_1/y_1 \ldots x_k/y_k))$ and the *output portion* $(out(x_1/y_1 \ldots x_k/y_k))$ of $x_1/y_1 \ldots x_k/y_k$ respectively.

The transition and output functions are extended to a sequence of inputs as follows, where ε denotes the empty sequence. For $\bar{x} \in X^\star$ and $x \in X$, $\delta(s, \varepsilon) = s$, $\delta(s, x\bar{x}) = \delta(\delta(s, x), \bar{x})$, $\lambda(s, \varepsilon) = \varepsilon$, $\lambda(s, x\bar{x}) = \lambda(s, x)\lambda(\delta(s, x), \bar{x})$.

An FSM can be represented by a directed graph. A vertex represents a state and a directed edge with label x/y that goes from a vertex with label s to a vertex with label s' represents the transition $\tau = (s, x, y, s')$.

Example 1. Figure 1 represents an FSM M_1 with state set $\{s_1, s_2, s_3, s_4\}$, inputs $\{x_1, x_2\}$, and outputs $\{y_1, y_2, y_3\}$.

The behaviour of an FSM M is defined in terms of the labels of walks that leave the initial state of M. A *walk* ω of M is a sequence of consecutive transitions $\omega = (s_1, x_1, y_1, s_2)(s_2, x_2, y_2, s_3) \ldots (s_{k-1}, x_{k-1}, y_{k-1}, s_k)(s_k, x_k, y_k, s_{k+1})$. Walk ω has *starting state* s_1, *ending state* s_{k+1}, and *label* $x_1/y_1x_2/y_2 \ldots x_k/y_k$. Here $x_1/y_1x_2/y_2 \ldots x_k/y_k$ is a *trace* of M.

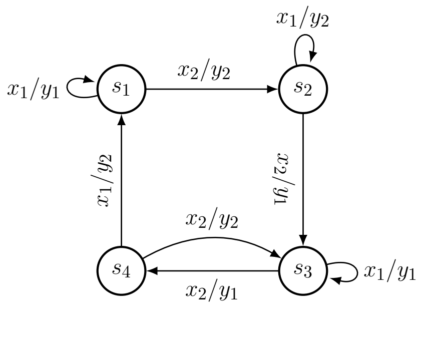


Fig. 1: An FSM M_1

Example 2. For example $\rho = (s_4, x_1, y_2, s_1)(s_1, x_1, y_1, s_1)(s_1, x_2, y_2, s_4)$ is a walk of M_1. The walk ρ has starting state s_4, ending state s_2, and label $x_1/y_2x_1/y_1x_2/y_2$. Here $x_1/y_2x_1/y_1x_2/y_2$ is a trace of M.

An FSM M defines the language L_M of labels of walks with starting state s_0 and we will use $L_M(s)$ to denote the language defined by making s the initial state of M. More formally, $L_M(s) = \{\bar{x}/\bar{y}|\bar{x} \in X^* \wedge \bar{y} = \lambda(s, \bar{x})\}$. Clearly, $L_M = L_M(s_0)$. Given $S' \subseteq S$, we let $L_M(S')$ denote the set of traces that can be produced if the initial state of M is in S', i.e., $L_M(S') = \cup_{s \in S'} L_M(s)$.

States s and s' of M are *equivalent* if $L_M(s) = L_M(s')$ and FSMs M and N are *equivalent* if $L_M = L_N$. FSM M is *minimal* if there is no equivalent FSM that has fewer states. FSM M is *strongly connected* if for every ordered pair (s, s') of states of M, there is a walk that has starting state s and ending state s'. Note that a strongly connected FSM M is minimal if and only if $L_M(s) \neq L_M(s')$ for all $s, s' \in S$ with $s \neq s'$. Throughout this paper we only consider minimal FSMs. This is not a significant restriction since one can convert an FSM into an equivalent minimal FSM in low order polynomial time [29].

Assumption 1 *We are testing from a minimal FSM $M = (S, s_0, X, Y, \delta, \lambda)$.*

Many test generation techniques use input sequences that identify states.

Definition 2. *An input sequence $\bar{x}$ defines a* unique input output sequence *for s if for all $s' \in S \setminus \{s\}$ we have that $\lambda(s, \bar{x}) \neq \lambda(s', \bar{x})$. Further, $\bar{x}$ defines a UIO for state set $S' \subseteq S$ if $\bar{x}$ defines a UIO for all $s \in S'$.*

3 Invertible sequences

In this section, we first define invertible sequences. We then discuss optimisation problems with potential impact on MBT related to invertible sequences.

3.1 Definitions

Due to their potential role in test generation, we are interested in walks that are invertible. A walk ρ with input/output label $\bar{x}/\bar{y}$ that has ending state s is an *invertible sequence* for s if no other walk with ending state s has label $\bar{x}/\bar{y}$.

For testing purposes, we may want to find a set of invertible sequences with a common input portion. Given a set $\varGamma$, of invertible sequences we use $\varGamma_i$ (respectively, $\varGamma_o$) to denote the set of input (respectively, output) portions of labels of the walks in $\varGamma$. We use $\varGamma_{in}$ (respectively, $\varGamma_{en}$) to denote the sets of initial (ending) states of walks in $\varGamma$. Let us suppose that S' is a set of states of M. Then we say that $\varGamma$ is an *invertible sequence for* S' if $\varGamma_i = \{\bar{x}\}$, $S' = \varGamma_{en}$, and all walks in $\varGamma$ are invertible sequences. An *invertible transition* is an invertible sequence of length one.

Let us assume that we are given an input sequence $\bar{x}$ that defines an invertible sequence for a set of states S'. Consider any partitioning of $\bar{x}$ as $\bar{x} = \bar{x}'\bar{x}''\bar{x}'''$ where $\bar{x}, \bar{x}', \bar{x}'', \bar{x}''' \in X^+$. If $\bar{x}'\bar{x}'''$ also defines an invertible sequence for S' then $\bar{x}$ is called a *redundant invertible sequence* for S'. In this paper, we consider only irredundant invertible sequences. If an invertible sequence is redundant, then it can be replaced by a shorter irredundant invertible sequence.

It has been shown that a suffix of an invertible sequence might not be an invertible sequence but a prefix is; this fact is formally state in the following lemma [27].

Lemma 1. *If $\rho = \rho'\rho''$ is an invertible sequence, then ρ' is an invertible sequence but ρ'' might not be an invertible sequence.*

We now define what it means for an invertible sequence to be proper. We say that invertible sequence ρ is a *proper invertible sequence* for s, if every suffix ρ' of ρ is also an invertible sequence for s. An immediate consequence of the definition of an invertible and an proper invertible sequence is that every proper invertible sequence is an invertible sequence but an invertible sequence need not be proper.

3.2 Invertible sequences in test generation

It has been shown that invertible sequences can be used to extend the set of UIOs [27].

Lemma 2. *(From [27]) If $\bar{x}/\bar{y}$ is a UIO for state s and $\rho = \bar{x}'/\bar{y}'$ is an invertible sequence for s starting from s' then $\bar{x}'\bar{x}/\bar{y}'\bar{y}$ is a UIO for s'.*

It should be noted that as every suffix of a proper invertible sequence ρ for s is a proper invertible sequence for s, a UIO for s can be used to compute a UIO for every state that a proper invertible sequence ρ visits.

Lemma 3. *Let $\bar{x}/\bar{y}$ be a UIO for state s, ρ be a proper invertible sequence for s and also let $\psi = \{(s', \rho') | s' \in S, \rho' \in post(\rho)$ and s' is the initial state of $\rho'\}$ be the set of pairs of suffixes of ρ and states from which they originate, then for each pair (s', ρ') in ψ, $in(\rho')\bar{x}/out(\rho')\bar{y}$ is a UIO for s'.*

This result suggests that in computing UIOs, longer proper invertible sequences are desirable, because longer invertible sequence lead to the derivation of more UIOs.[6] Therefore we investigated the following problem.

Definition 3. ***Longest proper invertible sequence (LPIS):*** *Let M be an FSM and also let s be a state of M. The LPIS problem is to decide whether there is a proper invertible sequence ρ for s such that $|in(\rho)| \geq K$.*

In the next section, we show that the LPIS problem is NP-complete.

Assume that for a given set of states S', we have computed a state identifying sequence and this time our aim is to derive state identification sequences for a specific set of states S'' without actually computing them. Due to Lemma 3, this can be achieved by using invertible sequences. However in order to reduce the memory/test cost spend on the test sequences, we want to compute a *preset* input sequence that takes S'' to S'. These requirements lead us to the following problem definition.

Definition 4. ***Preset reaching set invertible sequence (PRSIS):*** *Let M be an FSM and also let S' and S'' be sets of states of M of cardinality K. The PRSIS problem is to decide whether there are invertible sequences with common input portion $\bar{x}$ for S' such that $\bar{x}$ takes S'' to S'.*

In the next section we show that the PRSIS problem is PSPACE-complete.

The following problem is also motivated by the fact that in some cases we want to derive as many state identification sequences as possible from those already computed. In other words, we would like to find a set of invertible sequences to derive state identification sequences. However, considering the similar motivation as PRSIS problem, we are looking for invertible sequences with a minimum number of input portions.[7]

Definition 5. ***Minimum spanning invertible sequence (MINSIS):*** *Let M be an FSM and also let S' be a set of states of M. The MINSIS problem is to decide whether there is a set Γ of invertible sequences for S' where $|\Gamma_i| \leq K$ such that for all $s \in S \setminus S'$ there exists an invertible sequence in Γ that takes s to a state $s' \in S'$.*

We show that the MINSIS problem is NP-complete.

4 Complexity results

We show that the LPIS problem is NP-complete by providing a polynomial time reduction from the longest path problem (LPP) [30] to the LPIS problem. An instance of the LPP can be defined as follows, where a path[8] ($\mathcal{P}$) is said to *visit* a vertex v, if v is the starting vertex or the ending vertex of an edge in the path and the length of a path is the number of edges in the path.

[6] Recall that we restrict attention to invertible sequences that are not redundant.

[7] Recall that Γ_i is the set of input portions of labels of the walks in Γ.

[8] A path is a sequence of consecutive edges that, between them, do not visit any vertex more than once.

Definition 6. ***Longest path problem (LPP)*** *Consider a strongly connected directed graph $G = (V, E)$ with vertex set $V = \{v_1, v_2, \ldots, v_n\}$, edge set $E = \{e_1, e_2, \ldots, e_m\}$ and a positive integer $K < n$. The longest path problem for (G, K) is to decide whether there exists a path of G that visits at least K vertices.*

Let $out(v)$ be the number of outgoing edges of a vertex v. We let the *out-degree* ($Out(G)$) of the graph G be the maximum value of $out(v)$ for G i.e., $Max(\{out(v)|v \in V\})$.

Given an instance of the LPP (G, K), we construct an FSM $M(G) = (S, s_0, X, Y, \delta, \lambda)$. Our aim is to arrange the transition structure of $M(G)$ in such a way that an invertible sequence of length K defines a solution to the LPP. We now show how we construct $M(G)$.

For each vertex of G we introduce a corresponding state of $M(G)$ and we copy over the edge structure; if there is an edge from vertex v, represented by state s, to vertex v', represented by state s', then there is a transition from s to s'. We also introduce an additional special state $s_\star$. Then for each transition, we assign a unique integer i in the range $[1, |E|]$ and use it as the output label (y_i) of the corresponding transition in $M(G)$. In other words, the label of each transition in $M(G)$ will have a unique output portion.

The cardinality of the input alphabet of $M(G)$ is $Out(G)$, i.e., $X = \{x_1, x_2, \ldots, x_{Out(G)}\}$, for some arbitrary, yet pairwise distinct, $x_1, x_2, \ldots, x_{Out(G)}$. If s is a state of FSM $M(G)$ and the number of outgoing transitions is ℓ, then for each transition leaving s, we pick a unique element from the first ℓ elements of X (i.e., we pick an element from $\{x_1, x_2, \ldots, x_\ell\}$) and assign this symbol as the input label of the corresponding transition. Note that different states may have different numbers of outgoing edges, therefore the constructed $M(G)$ could be partial. We complete the missing transitions of state s_i by adding transitions to $s_\star$ with output y_i. We introduce a distinct input symbol $\star$ such that from every state s_i of $M(G)$, there exists a transition to $s_\star$ with common output y_i (see Figure 2). Finally, all transitions from $s_\star$ are self-loop transitions with output 0.

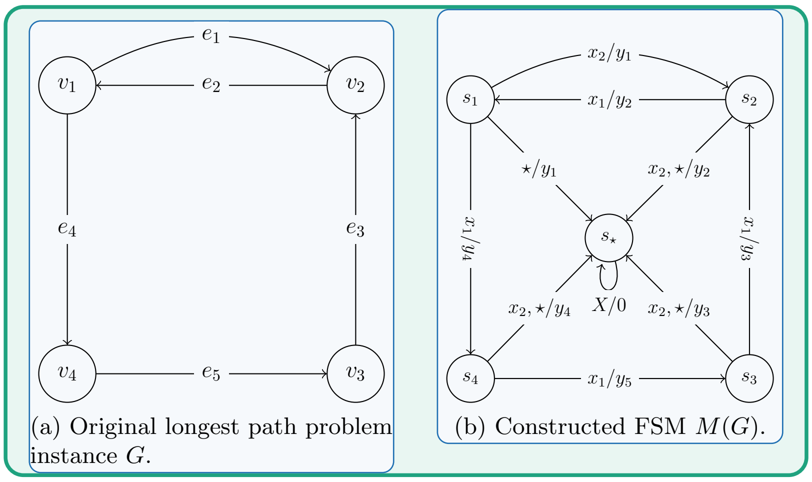
We now show how the longest path for a connected graph G relates to the LPIS problem for $M(G)$.

Proposition 1. *The longest path problem instance (G, K) has a solution if and only if state $s_\star$ of $M(G)$ has a proper invertible sequence ρ of length $K + 1$.*

Theorem 1. *The LPIS problem is* NP-complete*.*

We now show that MINSIS problem is NP-complete by a reduction from the minimum covering problem (MCP) [30].

Definition 7. ***Minimum covering problem (MCP)*** *Consider a set of elements $U = \{1, 2, \ldots, u\}$, a set of sets of elements $\mathcal{I} = \{I_1, I_2, \ldots, I_{\mathcal{I}}\}$ ($I_i \subseteq U$ for all $1 \leq i \leq \mathcal{I}$), and an integer K. The minimum covering problem is to decide whether there is a subset of $\mathcal{I}$ that contains at most K sets whose union is U.*



(a) Original longest path problem instance G.

(b) Constructed FSM $M(G)$.

Fig. 2: Construction of an FSM from a given longest path problem instance.

We show how FSM $M(U, \mathcal{I}, K)$ can be constructed such that the MCP problem for (U, I, K) corresponds to the MINSIS problem for $M(U, \mathcal{I}, K)$. For every $I_i \in U$, we introduce a single state s_i and, in addition, we introduce a special state $s^\star$. For every set I_j in $\mathcal{I}$, we introduce an input symbol x_j and an output symbol y_j. We also introduce a distinct output 0. The transition and output functions of $M(U, \mathcal{I}, K)$ are then defined as follows:

$$\delta(s_i, x_j) = \begin{cases} s_\star, & \text{if } i \in I_j \\ s_i, & \text{otherwise} \end{cases}$$

$$\lambda(s_i, x_j) = \begin{cases} y_i, & \text{if } i \in I_j \\ 0, & \text{otherwise} \end{cases}$$

The construction ends by setting $S' = \{s_\star\}$. Please see Figure 3 for an example.

Proposition 2. *The minimum covering problem instance $(G, \mathcal{I}, K)$ has a solution if and only if $S' = \{s_\star\}$ of $M(U, \mathcal{I}, K)$ has a minimum spanning invertible sequence $\mathit{\Gamma}$ with $|\mathit{\Gamma}_i| \leq K$.*

Theorem 2. *The MINSIS problem is* NP-complete.

We show that the PRSIS problem is PSPACE-complete by a reduction from the finite automata intersection problem (FA INT), which was introduced by Kozen [31]. In the FA INT problem we are given a set of *regular* automata with a common alphabet and our aim is to decide whether the automata accept a common word. A regular automaton is defined as follows.

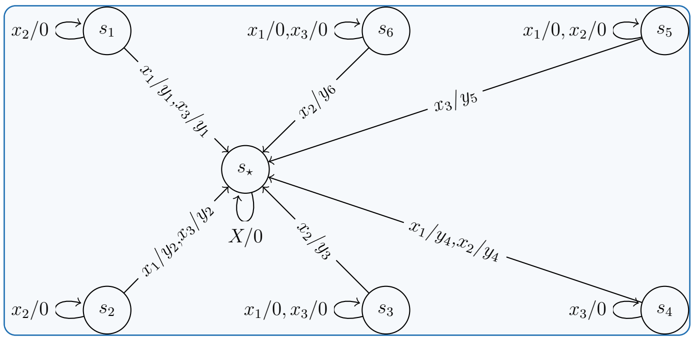


Fig. 3: Construction of a FSM $M(U, \mathcal{I}, K)$ from a given minimum covering problem instance $U = \{1, 2, 3, 4, 5, 6\}$, $\mathcal{I} = \{\{1, 2, 4\}, \{3, 4, 6\}, \{1, 2, 5\}\}$ and $K = 2$.

Definition 8. *A regular automaton is defined by 5-tuple $A = (Q, \Sigma, h, 0_A, F)$ where Q, Σ, h are a finite set of states, a finite set of inputs and a transition function, respectively. $0_A \in Q$ is the* initial state *and $F \subseteq Q$ is the set of* accepting state*. Automaton A* accepts *a word $w \in \Sigma^\star$ if $h(0_A, w) \in F$.*

Definition 9. *Let $\mathbb{A} = \{A_1, A_2, \ldots, A_z\}$ be a set of regular automata with a common alphabet Σ. The FA INT problem is to determine whether there is a word w such that $w \in L(A_i)$ for all $1 \leq i \leq z$.*[9]

We show that the PRSIS problem is PSPACE-complete. We first show how we construct an FSM from a given instance of the FA INT problem.

Without loss of generality, we assume that the finite automata in $\mathbb{A}$ have disjoint sets of states. Given an instance of the FA INT problem defined by set $\mathbb{A} = \{A_1, A_2, \ldots, A_z\}$ of finite automata on common finite alphabet Σ ($A_i = (Q_i, \Sigma, h_i, 0_i, F_i)$), we construct an FSM $M(\mathbb{A}) = (S, s_0, X, Y, \delta, \lambda)$ as follows.

We copy the states of each automaton $A_i = (Q_i, \Sigma, \delta_i, 0_i, C_i)$ and given $q_j \in Q_i$ we let s_j denote the corresponding state in S. For each A_i we also introduce an additional state $\star_i$. The input alphabet of the FSM is given by $X = \Sigma \cup \{f, f'\}$ and the output alphabet of the FSM is given by $Y = \{0, 1, 2, \ldots, z\}$. The state transitions of the finite automata in $\mathbb{A}$ are inherited: if $a \in \Sigma$ and $q_j \in Q_i$ for $1 \leq i \leq z$ and $1 \leq j \leq |Q_i|$ then $\delta(s_j, a) = s_k$ if $h_i(q_j, a) = q_k$. In a state of the form $\star_i$, an input from Σ leads to no change in state and output 0.

Each transition with input $x \in \Sigma$ produces output 0. For each $\star_i$, we introduce a transition from $\star_i$ to 0_i with label f/i; all other transitions with input f have output 0. We also introduce states $s_1^F, s_2^F, \ldots, s_z^F$ and input f'; the input of f' in a state from F_i leads to state s_i^F and the input of f' when the FSM is in a state from some $Q_i \setminus F_i$ leads to state $\star_1$. The input of f' always leads to output 0.

[9] Note that in some cases the initial state of each automaton is an accepting state. Clearly, for such cases an empty input sequence defines a solution to the FA INT problem instance, hence we do not consider such cases.

Finally we set $S'' = \{\star_1, \star_2, \dots, \star_z\}$ and $S' = \{s_1^F, s_2^F, \dots, s_z^F\}$.

Theorem 3. *PRSIS problem is* PSPACE-complete.

5 Conclusion

Many algorithms for generating test sequences from FSMs use UIOs but UIO existence is PSPACE-complete. As a result, UIO generation algorithms take advantage of situations in which one can generate additional UIOs from a UIO that has been found. The main such approach is to use invertible sequences [27, 28].

This paper has explored three optimisation problems associated with invertible sequences: deciding whether there is a (proper) invertible sequence of length at least K; deciding whether there is a set of invertible sequences, for state set S', that contains at most K input sequences; and deciding whether there is a single input sequence that defines invertible sequences that take state set S'' to state set S'. We proved that the first two problems are NP-complete and the third is PSPACE-complete.

There are several lines of future work. First, in practice we might have an upper bound on the length of an invertible sequence that is of interest; there is the problem of deciding whether the complexity results change if one incorporates such an upper bound. It would also be interesting to use experiments to explore properties of invertible sequences and UIOs. Finally, there is potential to use invertible sequences in generating other types of tests that distinguish states of an FSM. One might, for example, consider problems associated with generating adaptive distinguishing sequences for an FSM or a given set of states of an FSM.

Acknowledgments

This work is supported by the EU COST Action under Grant #IC1405 .

References

1. E. P. Moore. Gedanken-experiments. In C. Shannon and J. McCarthy, editors, *Automata Studies.* Princeton University Press, 1956.
2. F. C. Hennie. Fault-detecting experiments for sequential circuits. In *Proceedings of Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, pages 95–110, Princeton, New Jersey, November 1964.
3. 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 Transactions on Communications*, 39(11):1604 –1615, nov 1991.
4. T. S. Chow. Testing software design modelled by finite state machines. *IEEE Transactions on Software Engineering*, 4:178–187, 1978.
5. R. M. Hierons and H. Ural. Generating a checking sequence with a minimum number of reset transitions. *Automated Software Engineering*, 17(3):217–250, 2010.

6. H. Ural and K. Zhu. Optimal length test sequence generation using distinguishing sequences. *IEEE/ACM Transactions on Networking*, 1(3):358–371, 1993.
7. G. L. Luo, G. v. Bochmann, and A. Petrenko. Test selection based on communicating nondeterministic finite-state machines using a generalized Wp-method. *IEEE Transactions on Software Engineering*, 20(2):149–161, 1994.
8. A. Petrenko, N. Yevtushenko, and G. v. Bochmann. Testing deterministic implementations from nondeterministic FSM specifications. In *IFIP TC6 9th International Workshop on Testing of Communicating Systems*, pages 125–141, Darmstadt, Germany, 9–11 September 1996. Chapman and Hall.
9. A. Gill. *Introduction to The Theory of Finite State Machines.* McGraw-Hill, New York, 1962.
10. Z. Kohavi. *Switching and Finite State Automata Theory.* McGraw-Hill, New York, 1978.
11. R. T. Boute. Distinguishing sets for optimal state identification in checking experiments. *IEEE Transactions on Computers*, 23:874–877, 1974.
12. H. Ural, X. Wu, and F. Zhang. On minimizing the lengths of checking sequences. *IEEE Transactions on Computers*, 46(1):93–99, 1997.
13. 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. In *Protocol Specification, Testing, and Verification VIII*, pages 75–86, Atlantic City, 1988. Elsevier (North-Holland).
14. W. Y. L. Chan, C. T. Vuong, and M. R. Otp. An improved protocol test generation procedure based on UIOs. *SIGCOMM Computer Communication Review*, 19(4):283–294, August 1989.
15. W.-H. Chen and H. Ural. Synchronizable test sequences based on multiple UIO sequence. *IEEE/ACM Transactions on Networking*, 3(2):152–157, 1995.
16. S. Guyot and H. Ural. Synchronizable checking sequences based on UIO sequences. In *Protocol Test Systems, VIII*, pages 385–397, Evry, France, September 1995. Chapman and Hall.
17. H. Motteler, A. Chung, and D. Sidhu. Fault coverage of UIO-based methods for protocol testing. In *Proceedings of Protocol Test Systems VI*, pages 21–33, 1994.
18. T. Ramalingam, K. Thulasiraman, and A. Das. A generalization of the multiple UIO method of test sequence selection for protocols represented in FSM. In *The 7th International workshop on Protocol Test Systems*, pages 209–224, Japan, 1994. Chapman and Hall.
19. H. Ural and Z. Wang. Synchronizable test sequence generation using UIO sequences. *Computer Communications*, 16(10):653–661, 1993.
20. S. T. Vuong, W. W. L. Chan, and M. R. Ito. The UIOv-method for protocol test sequence generation. In *The 2nd International Workshop on Protocol Test Systems*, Berlin, 1989.
21. D. Lee and M. Yannakakis. Testing finite-state machines: State identification and verification. *IEEE Transactions on Computers*, 43(3):306–320, 1994.
22. M. P. Vasilevskii. Failure diagnosis of automata. *Cybernetics*, 4:653–665, 1973.
23. R. M. Hierons and U. C. Türker. Parallel algorithms for generating harmonised state identifiers and characterising sets, (accepted). *IEEE Transactions on Software Engineering*, –(–):–, 2016.
24. G. Gonenc. A method for the design of fault detection experiments. *IEEE Transactions on Computers*, 19:551–558, 1970.
25. R. M. Hierons and H. Ural. Optimizing the length of checking sequences. *IEEE Transactions on Computers*, 55:618–629, May 2006.

26. R. M. Hierons and H. Ural. Reduced length checking sequences. *IEEE Transactions on Computers*, 51(9):1111–1117, 2002.
27. R. M. Hierons. Extending test sequence overlap by invertibility. *The Computer Journal*, 39(4):325–330, 1996.
28. K. Naik. Efficient computation of unique input/output sequences in finite-state machines. *IEEE/ACM Transactions on Networking*, 5(4):585–599, August 1997.
29. J. E. Hopcroft. An n log n algorithm for minimizing the states in a finite automaton. In Z. Kohavi, editor, *The theory of Machines and Computation*, pages 189–196. Academic Press, 1971.
30. M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman and Company, New York, 1979.
31. D. Kozen. Lower bounds for natural proof systems. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 254–266, Washington, DC, USA, 1977. IEEE Computer Society.
32. Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177 – 192, 1970.

Appendix

Lemma 3. *Let $\bar{x}/\bar{y}$ be a UIO for state s, ρ be a proper invertible sequence for s and also let $\psi = \{(s', \rho') | s' \in S, \rho' \in post(\rho)$ and s' is the initial state of $\rho'\}$ be the set of pairs of suffixes of ρ and states from which they originate, then for each pair (s', ρ') in ψ, $in(\rho')\bar{x}/out(\rho')\bar{y}$ is a UIO for s'.*

Proof. We use proof by contradiction. Let ψ be the set of pairs of suffixes and states of some invertible sequence ρ for state s. Consider a pair (s', ρ') and let us suppose that $in(\rho')\bar{x}/out(\rho')\bar{y}$ is not a UIO for s'. This implies that there exists a state $s'' \neq s'$ such that there exists a walk from s'' labeled with input/output sequence $in(\rho')\bar{x}/out(\rho')\bar{y}$. Now consider the state s''' reached from s'' with walk $in(\rho')/out(\rho')$. As the underlying FSM is deterministic we have two options:

- we have $s''' = s$,
- or we have $s''' \in S \setminus \{s\}$.

In the first case, ρ' cannot be an invertible sequence. Otherwise, if the second case holds, then $\bar{x}/\bar{y}$ cannot be a UIO for s. The result thus follows.

□

Proposition 1. *The longest path problem instance (G, K) has a solution if and only if state $s_\star$ of $M(G)$ has a proper invertible sequence ρ of length $K + 1$.*

Proof. First we prove that if G has a path $\mathcal{P} = e_1 e_2 \dots e_K$ of length K, then $M(G)$ has a proper invertible sequence for $s_\star$ whose input portion has length $K + 1$. First note that for every vertex and edge of G there exists a state and a transition in $M(G)$ respectively. Let $\rho = x_1/y_1 x_2/y_2 \dots x_K/y_K$ be the label of the walk corresponding to $\mathcal{P}$. Since every transition of $M(G)$ is labelled with unique input/output values, $\rho = x_1/y_1 x_2/y_2 \dots x_K/y_K$ defines an invertible sequence for a state of $M(G)$. Finally, if we concatenate ρ with some $\rho' = \star/y_j$, which is the label of a walk that starts from the ending state of walk ρ, then $\rho'' = \rho\rho'$ defines an invertible sequence for $s_\star$.

Now assume that $s_\star$ has a proper invertible sequence $\rho = x_1/y_1 x_2/y_2 \dots x_{K+1}/y_{K+1}$ of length $K + 1$ and we are required to prove that G has a path of length K. Note that since ρ is an invertible sequence for $s_\star$, the last input/output pair belongs to a transition that takes $M(G)$ to state $s_\star$. Besides, since ρ is a proper invertible sequence, the first K symbols of the input portion of ρ should visit $K+1$ different states of $M(G)$. Since for every state and transition of $M(G)$, there exists a corresponding vertex and edge in G, the first K inputs of ρ define a path of G with length K. Thus the result follows.

□

Theorem 1. *The LPIS problem is* NP-complete.

Proof. We first show that the LPIS problem is in NP. A non-deterministic Turing machine can guess an input sequence $\bar{x}$ of length K. It can then apply $\bar{x}$ to every state and record the resultant output sequence and state reached. Afterwards, it can compare the outputs to decide whether $\bar{x}$ defines an invertible sequence for a specific state s.

The problem is NP-hard due to Proposition 1 and the fact that the longest path problem with directed graphs is NP-hard. Therefore the result follows.

□

Proposition 2. *The minimum covering problem instance $(G, \mathcal{I}, K)$ has a solution if and only if $S' = \{s_\star\}$ of $M(U, \mathcal{I}, K)$ has a minimum spanning invertible sequence Γ with $|\Gamma_i| \leq K$.*

Proof. First we prove that if $U, \mathcal{I}, K$ has a minimum covering $\mathcal{I}' = \{I_1, I_2, \ldots, I_K\}$ then $M(U, \mathcal{I}, K)$ has a set of invertible sequences Γ for $S' = \{s_\star\}$ such that $\Gamma_i = \{x_1, x_2, \ldots, x_K\}$. Note that the transitions and output functions of the FSM $M(U, \mathcal{I}, K)$ dictates that for a given input x_i and output y_j pair, there exists at most one transition with ending state $s^\star$ and label x_i/y_j. Therefore, each transition with ending state $s_\star$ is an invertible transition and hence there is a set Γ of invertible sequences that take M from $S \setminus \{s_\star\}$ to $s_\star$. Further, for every set I_i in $\mathcal{I}$ there exists a single corresponding input symbol x_i and so $\Gamma_i = \{x_1, \ldots, x_K\}$. Thus, Γ defines a spanning invertible sequence for S' with $|\Gamma_i| = K$ as required.

Now we assume that $S' = \{s_\star\}$ has a maximum spanning invertible sequence Γ such that $|\Gamma_i| = K$ and we are required to prove that U has a minimum covering with at most K sets. First note that as we only consider invertible sequences that are not redundant, the length of each input sequence in set Γ_i is one. Let $\Gamma_i = \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_K\}$. Therefore, there is a set $\mathcal{I}' = \{I_1, I_2, \ldots, I_K\}$ of sets derived from Γ_I. The result thus follows.

□

Theorem 2. *The MINSIS problem is* NP-complete*.*

Proof. The proof of being in NP is almost similar to that of Theorem 1. However this time Turing machine should guess at most K input sequences. The problem is NP-harddue to Proposition 2, thus the result follows.

□

Theorem 3. *PRSIS problem is* PSPACE-complete*.*

Proof. We first show that the PRSIS problem is in PSPACE. The working principle of the Turing machine for the PRSIS problem is as follows. First note that a non-deterministic Turing machine $\mathcal{T}$ can take S'' to S' input by input as follows: Let w be the sequences of inputs guessed by $\mathcal{T}$ so far, and $\mathcal{T}$ guesses an input x. After this point $\mathcal{T}$ applies x to states $\delta(S'', w)$. $\mathcal{T}$ should then check whether a) $\delta(S'', wx) = S'$, and b) For all $s \in S''$ and $s' \in S$, if $\delta(s, wx) = \delta(s', wx)$ then $\lambda(s, wx) \neq \lambda(s', wx)$ If these three conditions hold $\mathcal{T}$ returns at accepting state. Otherwise it returns at rejecting state.

To achieve this $\mathcal{T}$ maintains (and updates on each iteration) the following information (given input sequence w): 1) a partition D of S'' saying which pairs of states from S'' are not distinguished by w. 2) For each state $s \in S''$, the pair (s', S''') where: $s' = \delta(s, w)$ is the current state corresponding to s and S''' is the set of current states from states in $S \setminus S''$ that are not distinguished from s. Thus $S''' = \{\delta(s'', w) | s'' \in S \setminus S'' \wedge \lambda(s, w) = \lambda(s'', w)\}$

Clearly, it is straightforward to update this information if we extend w to wx (guessing x). It is also easy to spot when one should not extend further by x (either the current states reached from states of S'' not distinguished by w 'converge' or there is some (s', S''') such that s' and a state from S''' 'converge').

The above can clearly be stored in polynomial space. In addition to the terminating conditions mentioned above, $\mathcal{T}$ should terminate when the upper bound is reached. First note that the number of possible values for a pair (s', S''') is bounded above by $n.2^n$ and so the number of possible such pairs is bounded above by: $(n2^n)^K = n^K.2^{nK}$. Second, initially D contains K sets. The only way we can change D is by merging two or more sets, with this reducing the number of sets in D. Thus, the value of D can change at most $K - 1$ times.

Therefore the upper bound for the PRSIS is $(K-1).n^K.2^{(nK)}$. Note that this information can be stored in polynomial space, i.e. $O(log((K-1).n^K.2^{(nK)})) = O(log(K-1) + Klog(n) + nKlog(2))$ space and the Turing machine $\mathcal{T}$ can hold a counter and increment this by one after an input is guessed. Therefore when the value stored in the counter exceeds the upper bound value, $\mathcal{T}$ terminates.

Therefore, the entire search in this way can be performed in NPSPACE. Based on Savitch's Theorem [32], the PRSIS problem is in PSPACE as required.

Now we prove that if the automata accept a common word $w \in \Sigma$ then $M(\mathbb{A})$ has an invertible sequence that takes S'' to S'. Clearly the application of fwf' from a state of S'' brings $M(\mathbb{A})$ to one of states in S'. As the output produced as a response to input f is unique, fwf' is a PRSIS for S' as required.

Now we assume that there are invertible sequences with common input sequence $\bar{x}$ that take S'' to S' and we are required to prove that there is a common element for the automata in $\mathbb{A}$. Note since $\bar{x}$ takes S'' to S', the input sequence $\bar{x}$ should contain at least one f and must end with f'. Let $\bar{x}'f'$ be the suffix of $\bar{x}$ after the first input f. After the application of f, the FSM is in a state that corresponds to an initial state of the corresponding automaton. Since $\bar{x}$ takes S'' to S', $\bar{x}'f'$ must takes set $\delta(S'', f)$ to S' and so $\bar{x}$ must take initial states of the A_i to final states. The result thus follows setting $w = \bar{x}$.

□