Generating a Checking Sequence with a minimum number of reset transitions

R. M. Hierons · H. Ural

Received: date / Accepted: date

Abstract Given a finite state machine M, a checking sequence is an input sequence that is guaranteed to lead to a failure if the implementation under test is faulty and has no more states than M. There has been much interest in the automated generation of a short checking sequence from a finite state machine. However, such sequences can contain reset transitions whose use can adversely affect both the cost of applying the checking sequence and the effectiveness of the checking sequence. Thus, we sometimes want a checking sequence with a minimum number of reset transitions rather than a shortest checking sequence. This paper describes a new algorithm for generating a checking sequence, based on a distinguishing sequence, that minimises the number of reset transitions used.

Keywords Finite state machine · checking sequence generation · reset transition · distinguishing sequence · optimisation

1 Introduction

The importance and cost of testing has led to much interest in automated test generation. Automation is facilitated by the presence of a model or a formal specification that describes the required behaviour of the implementation under test (IUT). State-based systems are often specified or modelled using finite state machines (FSMs) or languages such as Statecharts [13] and SDL [20] that are based on extended finite state machines

This work was supported in part by Leverhulme Trust grant number F/00275/D, Testing State Based Systems, Natural Sciences and Engineering Research Council (NSERC) of Canada grant number RGPIN 976, and Engineering and Physical Sciences Research Council grant number GR/R43150, Formal Methods and Testing (FORTEST).

R. M. Hierons
Department of Information Systems and Computing, Brunel University, Uxbridge, Middlesex, UB8 3PH, UK
E-mail: rob.hierons@brunel.ac.uk

H. Ural
School of Information Technology and Engineering, University of Ottawa, Ottawa, Ontario, K1N 6N5, Canada

(EFSMs). When testing from an EFSM it is common to produce a corresponding FSM by either abstracting out the internal data or expanding this out, possibly after putting limits on this data. FSM based test generation techniques can then be applied. If an abstraction is used then it is possible to choose paths that are feasible in the abstraction and infeasible in the original EFSM, but this problem has been solved for certain classes of EFSM [8]. Since state-based systems can be represented using FSMs there has been much interest in the problem of automatically generating a test sequence from an FSM [5,15,17,18,25].

FSM based test sequence generation has received attention in several domains. It is normal to specify communications protocols and embedded systems using state-based languages and here FSM based techniques are applicable [4,21]. The use of FSM based techniques has also been proposed in the testing of object-oriented systems [3], web services [14] and model-based testing [2,9]. It transpires that FSM based test generation techniques assist testing from a specification in a formal language such as Z, VDM or B [7].

It is normal to use a criterion that states what it means for a test sequence to be adequate. One criterion is that the test sequence is a *checking sequence*: it is guaranteed to determine correctness as long as the IUT has no more states than the specification FSM. The notion of a checking sequence was introduced by Moore [24] and Hennie showed how a checking sequence can be produced when the FSM has a known distinguishing sequence[1] [15]. Hennie represented checking sequence generation in terms of testing the transitions of an FSM but also showed that there are other types of checking sequence. Since Hennie's paper, research in this area has focussed on the problem of producing a short checking sequence for an FSM that has a known distinguishing sequence (see, for example, [12,17,18,25]). The resultant checking sequence generation algorithms are based on a sufficient condition for an input sequence to be a checking sequence, the sufficient condition requiring that each transition is tested, and aim to produce a shortest input sequence that satisfies this condition.

This paper considers the testing of a resetable IUT: one that has a reset operation that is known to (correctly) return the IUT to its initial state. The transitions triggered by this reset operation are known as *reset transitions.* The reset of a system can require the reconfiguration of the system and can involve human actions and so each use of a reset transition significantly *increases the cost* of testing [10,11,16,27]. If a fault in a system is associated with there being extra states then we may require long checking sequences in order to detect this fault [4,10,11]. However, reset transitions split a checking sequence into separate subsequences and so reduce the chance of finding such faults: they *reduce the effectiveness* of a checking sequence. Since a reset transition can significantly increase the cost of applying a checking sequence and reduce the effectiveness of a checking sequence, for some applications we wish to produce a checking sequence with a minimum number of reset transitions. This paper adapts a class of algorithms for producing a checking sequence [17,18,25] so that we get a checking sequence that, amongst those that can be produced by this class of algorithms, has the fewest reset transitions. Such a checking sequence is said to be *optimal.* In contrast to other algorithms for generating a checking sequence, the proposed algorithm does not require the FSM to be strongly connected. The use of adaptive checking sequences may well provide additional benefits, and in particular the use of adaptive distinguishing se-

[1] A distinguishing sequence for an FSM M is an input sequence that leads to different output sequences for the different states of M. Distinguishing sequences are defined in Section 2

quences [21]. However, this is left as a topic for future work. There has also been recent work that shows how short checking sequences can be produced using weaker sufficient conditions for an input sequence to be a checking sequence [6]. However, this work produces many separate subsequences separated by resets and so is not appropriate when we wish to minimise the number of resets.

The checking sequence generation algorithms described in [17,18,25] operate in the following manner. First, they produce two sets of walks from the digraph $G(M)$ that represents the FSM M: a set E_α of walks that check that the distinguishing sequence $\bar{D}$ used works correctly in the IUT and a set E_t of walks that use $\bar{D}$ to test the transitions. For each walk in $E_t \cup E_\alpha$ an edge is added to $G(M)$ and this produces a digraph $TestG(M)$. A minimum cost walk of $TestG(M)$ that includes each edge in $E_t \cup E_\alpha$ is produced and a checking sequence generated from this. This walk is devised using two steps. In the first step, a minimum number of copies of edges from $TestG(M)$ are added to $E_t \cup E_\alpha$ in order to make the resultant digraph $Aug(M)$ symmetric: for each vertex v of $Aug(M)$ there are the same number of edges that enter v as leave v. If $Aug(M)$ is connected then the checking sequence is produced from this. Otherwise walks are added to connect the components of $Aug(M)$ and a checking sequence is generated. This approach chooses the set of edges and walks, added to $E_t \cup E_\alpha$, in a manner that guarantees that additions are acyclic as this is required in order to satisfy the sufficient condition from [25] for an input sequence to be a checking sequence.

The algorithm given in this paper adapts this approach in several ways. First, the edges in $G(M)$ that represent resets are given a cost that is sufficiently high to ensure that a minimum cost walk that contains every edge in $E_t \cup E_\alpha$ also minimises the number of resets. Again, a walk is produced through two steps. First, a digraph $TestG(M)$ is defined and a minimum cost set of copies of edges from $TestG(M)$ is added to $E_t \cup E_\alpha$ in order to produce a symmetric digraph $Aug(M)$. If $Aug(M)$ is connected then we produce a checking sequence and it is guaranteed that this minimises the number of resets. If $Aug(M)$ is not connected then we need to add walks to connect it but we wish to do so in a way that adds as few resets as possible. In this paper we prove that the sufficient condition from [25], for an input sequence to be a checking sequence, can be weakened and that we can always add a set of walks with no resets that connect $Aug(M)$ and does not invalidate the new sufficient condition. As a result, the step that connects the components of $Aug(M)$ adds no resets and so the resultant checking sequence minimises the number of resets. Interestingly, while the checking sequence generation algorithms in [17,18,25] require an NP-hard optimisation problem to be solved if we want to be guaranteed to return a shortest checking sequence, the algorithm given in this paper minimises the number of resets and has low order polynomial time complexity.

The rest of the paper is organised as follows. Section 2 describes finite state machines and digraphs. Section 3 gives a sufficient condition, for a test sequence to be a checking sequence, that forms the basis of checking sequence generations. Section 4 gives the algorithm for generating a checking sequence and Section 5 reports the results of experiments. Finally, Section 6 draws conclusions.

2 Preliminaries

2.1 Notation

Throughout this paper ϵ denotes the empty sequence. We will put a bar above the name of a variable (e.g. $\bar{x}$) if this variable represents a sequence. Given a sequence $\bar{\rho} = z_1, \ldots, z_k$, for all $1 \leq i \leq k$ we have that $z_1, \ldots, z_i$ is a *prefix* of $\bar{\rho}$ and $z_i, \ldots, z_k$ is a *suffix* of $\bar{\rho}$.

2.2 Directed Graphs

Given a set L of labels, a *directed graph* (*digraph*) G is defined by a pair (V, E) where V is a set of *vertices* and $E \subseteq V \times V \times L$ is a set of directed *edges* between the vertices. Given edge $e = (v, v', l)$, v is the *starting vertex* of e, v' is the *ending vertex* of e, and l is the *label* of e. G is *symmetric* if for every vertex v of V the number of edges whose starting vertex is v is equal to the number of edges whose ending vertex is v.

The test generation algorithm proposed in this paper will require us to use digraphs in which there can be more than one copy of an edge. In such situations we use a multiset of edges rather than a set of edges. Multisets differ from sets in one important way: each element of a multiset occurs a specified number of times in that multiset. Thus, if E' is a multiset of elements of set E then we can represent E' as a set of pairs of the form (e, k) where $e \in E$ and k is the number of times that e occurs in E'.

A *walk* in G is a sequence $e_1, \ldots, e_m$ of successive pairs of adjacent edges from G. If $e_i = (v_i, v_{i+1}, l_i)$ for all $1 \leq i \leq m$ then $e_1, \ldots, e_m$ has *label* $l_1, \ldots, l_m$, *starting vertex* v_1, and *ending vertex* v_{m+1}. Given a walk $\bar{P} = e_1, \ldots, e_m$, for all $1 \leq i \leq j \leq m$ we have that $e_i, \ldots, e_j$ is a *subwalk* of $\bar{P}$. If we assign a cost *cost*(e) to each edge e of G then the cost of a walk is the sum of the costs of the edges in the walk: *cost*$(e_1, \ldots, e_m) = \sum_{1 \leq i \leq m}$ *cost*(e_i).

G is *strongly connected* if for every ordered pair (v, v') of vertices of G there is a walk from v to v'. G is *weakly connected* if the underlying undirected graph is connected: for every ordered pair of vertices (v, v') of G there is a sequence $(v_1, v_2, l_1), \ldots, (v_k, v_{k+1}, l_k)$ with $v = v_1$ and $v' = v_{k+1}$ such that for all $1 \leq i \leq k$ we have that either (v_i, v_{i+1}, l_i) is an edge of G or (v_{i+1}, v_i, l_i) is an edge of G. A *path* is a walk in which no vertex is repeated and a walk $e_1, \ldots, e_m$ is a cycle if $e_1, \ldots, e_{m-1}$ is a path and the ending vertex of e_m is the starting vertex of e_1. If $\bar{P}_1$ is a subwalk of a path $\bar{P}$ then $\bar{P}_1$ is a *subpath* of $\bar{P}$. Walk $e_1, \ldots, e_m$ is a *tour* if its starting and ending vertices are the same. If we *start* tour $e_1, \ldots, e_m$ with edge e_i we get $e_i, \ldots, e_m, e_1, \ldots, e_{i-1}$. A tour of G is an *Euler Tour* if it contains each edge of G exactly once.

Given digraph $G = (V, E)$ and a set $E' \subseteq E$ of edges, the rural Chinese postman problem (RCPP) is to find a shortest tour of G that contains every edge from E'. While the RCPP is NP-hard [22], a polynomial time heuristic is often applied when test sequence generation is represented in terms of the RCPP. In this heuristic [1], we first find a minimum cost symmetric augmentation of E': a minimum cost symmetric multiset of elements of E that contains E'. This is a multiset since it may be necessary to include multiple copies of some edges. If the resultant digraph is strongly connected then an Euler Tour of this digraph is a solution to the RCPP; if the digraph isn't strongly connected then we add edges to connect its components and the resultant tour may be suboptimal.

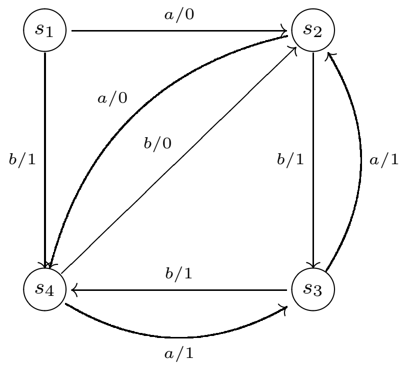

Fig. 1 Finite State Machine M_0

Given digraph $G = (V, E)$, digraph $G' = (V', E')$ is a *subgraph* of G if $V' \subseteq V$ and $E' \subseteq E$. Given $E' \subseteq E$, $G[E']$ denotes the smallest subgraph of G that has edge set E'. Thus $G[E'] = (V', E')$ where V' is the set of vertices from V that are either starting vertices or ending vertices of edges from E': $V' = \{v \in V | \exists e = (v_i, v_j, l) \in E'.v = v_i \vee v = v_j\}$. A subgraph $G' = (V', E')$ of $G = (V, E)$ is a *component* of G if G' is strongly connected, there is no edge in G from a vertex in V' to a vertex not in V', and every edge e in E that is between vertices from V' is also in E': $E' = \{(v_i, v_j, l) \in E | v_i, v_j \in V'\}$.

2.3 Finite State Machines and Resets

A (completely specified and deterministic) FSM M is defined by a tuple $(S, X, Y, \delta, \lambda, s_1)$ in which S is the finite *set of states*; $s_1 \in S$ is the *initial state*; X is the finite *input alphabet*; Y is the finite *output alphabet*; δ is the *state transfer function* of type $S \times X \to S$; and λ is the *output function* of type $S \times X \to Y$. FSMs are sometimes called Mealy machines or finite state transducers. If we input $x \in X$ when M is in state $s \in S$ then we get output $y = \lambda(s, x)$ and M moves to state $s' = \delta(s, x)$. This defines a *transition* $(s, s', x/y)$ that has *starting state* s and *ending state* s'. The functions δ and λ can be extended to take input sequences in the usual way. Throughout this paper we assume that a completely specified and deterministic FSM $M = (S, X, Y, \delta, \lambda, s_1)$ with n states describes the required behaviour of the IUT. Consider the FSM M_0 in Figure 1. Here, for example, $\delta(s_2, a) = s_4$, $\lambda(s_2, a) = 0$ and so M_0 contains the transition $(s_2, s_4, a/0)$. If we consider the application of sequence ab from state s_2 we find that $\delta(s_2, ab) = s_2$ and $\lambda(s_2, ab) = 00$.

The FSM M can be represented by a digraph $G(M) = (V, E)$ in which each state s_i is represented by a corresponding vertex v_i, $|V| = n$, and $E = \{(v_i, v_j, x/y) | 1 \leq i, j \leq n \wedge x \in X \wedge \lambda(s_i, x) = y \wedge \delta(s_i, x) = s_j)$. As a result we can use graph theory terminology and notation when discussing FSMs.

If input sequence $\bar{x} = x_1, \ldots, x_k$ is applied when M is in state s_i then it takes M to state $s_j = \delta(s_i, \bar{x})$ and leads to output sequence $\bar{y} = y_1, \ldots, y_k = \lambda(s_i, \bar{x})$. This is achieved through a sequence $\bar{\rho} = \tau_1, \ldots, \tau_k$ of consecutive transitions of M and $\bar{\rho}$ is said to be a *transition sequence* whose *label* is $label(\bar{\rho}) = x_1/y_1, \ldots, x_k/y_k$. Transition sequence $\bar{\rho}$ has *starting state* s_i and *ending state* s_j. In addition, $x_1/y_1, \ldots, x_k/y_k$ is an *input/output sequence* and can be represented by $\bar{x}/\bar{y}$. We call $\bar{x}$ the *input portion* of $\bar{x}/\bar{y}$ while $\bar{y}$ is the *output portion* of $\bar{x}/\bar{y}$. The transition sequence $\bar{\rho}$ can be represented by the tuple $(s_i, s_j, x_1/y_1, \ldots, x_k/y_k)$ or by the tuple $(s_i, s_j, \bar{x}/\bar{y})$. In addition, this transition sequence is *represented* by an edge $(v_i, v_j, x_1/y_1, \ldots, x_k/y_k)$ that can be added to $G(M)$ when it is desired to construct an augmented version of $G(M)$.

The digraph $G(M)$ representing an FSM M is *initially connected* if each state s of M can be reached from s_1: for every state $s \in S$ there is an input sequence $\bar{x}$ such that $s = \delta(s_1, \bar{x})$. If $G(M)$ is not initially connected then the unreachable states can be deleted and thus only FSMs represented by initially connected digraphs will be considered. $G(M)$ is *strongly connected* if for each ordered pair of states (s, s') there is an input sequence $\bar{x}$ that takes M from s to s'.

States s and s' are *equivalent* if for every input sequence $\bar{x}$ we have that $\lambda(s, \bar{x}) = \lambda(s', \bar{x})$. If for an input sequence $\bar{x}$ we have that $\lambda(s, \bar{x}) \neq \lambda(s', \bar{x})$ then $\bar{x}$ *distinguishes* between s and s'. Input sequence $\bar{D}$ is a *distinguishing sequence* for M if $\bar{D}$ distinguishes between states s and s' for all $s, s' \in S$ with $s \neq s'$. For example, in M_0 we have that $\lambda(s_1, aa) = 00$, $\lambda(s_2, aa) = 01$, $\lambda(s_3, aa) = 10$, and $\lambda(s_4, aa) = 11$ and so aa is a distinguishing sequence. In this paper we assume that M has a known distinguishing sequence $\bar{D}$.

Two FSMs are *equivalent* if their initial states are equivalent. If FSMs M and M' are not equivalent, and $\bar{x}$ distinguishes between their initial states, then $\bar{x}$ *distinguishes* between M and M'. FSM M is *minimal* if there is no FSM equivalent to M that has fewer states than M. We assume that any FSM considered is minimal since for an FSM M with n states and p inputs an equivalent minimal FSM can be produced in $O(np \log n)$ [19][2]. If the FSM being considered is not minimal then we require an initial preprocessing phase that minimises it.

The IUT has a *reliable reset feature* if there is a process that is known to correctly take it from any state to its initial state. In this paper the use of such a process is represented by the input of r; r takes the IUT from any state to its initial state and produces no output. The transitions triggered by r are *reset transitions* and thus for every state s of M we add the reset transition $(s, s_1, r/-)$ where $-$ represents null output. In order to simplify the exposition we assume that r is not contained within the input alphabet X since there is no need to test the reset transitions; the reset transitions are implicit. As a result the digraph $G(M)$ that represents M does not include edges that represent the reset transitions and so we define the set $E_R = \{(v_i, v_1, r/-) | 1 \leq i \leq n\}$ of additional edges that represent these transitions. If we apply a reset then we lose all information about the state before this and thus reset transitions cannot assist in distinguishing states. We thus assume that the distinguishing sequence $\bar{D}$ used does not contain r. If $\bar{D}$ contains r then we can produce a shorter distinguishing sequence by deleting this instance of r from $\bar{D}$ and all inputs that are after r. This is because the response of M to the reset r and the inputs after r does not depend on the state in which we applied $\bar{D}$ and so provides no information about this state.

In testing it is normal to assume that the IUT behaves like an unknown FSM M_I from a given fault domain. One standard fault domain is the set Φ_M of FSMs with the

[2] While the minimisation algorithm was developed for finite automata, we can represent an FSM M as a finite automaton whose alphabet is the set of input/output pairs of M.

same input and output alphabets as M and no more states than M. An input sequence $\bar{x}$ is a *checking sequence* if it distinguishes between M and every FSM from Φ_M that is not equivalent to M. In this paper we assume that the IUT behaves like an unknown FSM $M_I \in \Phi_M$. An alternative that has been considered in the literature is to assume that the IUT has at most k more states than M for some value k that is chosen by the tester (see, for example, [5, 23, 26]). However, the size of the resultant test grows exponentially as k increases. In addition, such methods produce (exponentially) many test sequences and separate these using resets. As a result they are not suitable when trying to produce a test sequence that contains very few resets.

Let $\bar{P}$ denote a walk $e_1, \ldots, e_m$ in $G(M)$ with starting vertex v_1 and $\bar{Q} = \mathit{label}(\bar{P})$. In order to reason about the state of the IUT reached by a prefix of $\bar{Q}$, we will define a digraph $\mathit{Linear}(\bar{Q})$.

Definition 1 Given an input/output sequence $\bar{Q} = x_1/y_1, \ldots, x_m/y_m$ we let $\mathit{Linear}(\bar{Q}) = (V(\bar{Q}), E(\bar{Q}))$ where

1. $V(\bar{Q}) = \{n_1, \ldots, n_{m+1}\}$
2. $E(\bar{Q}) = \{(n_1, n_2, x_1/y_1), \ldots, (n_m, n_{m+1}, x_m/y_m)\}$.

The vertices of $\mathit{Linear}(\bar{Q})$ are called *nodes*.

Given an input/output sequence $\bar{Q} = x_1/y_1, \ldots, x_m/y_m$ and a subsequence $\bar{Q}' = x_i/y_i, \ldots, x_j/y_j$ of $\bar{Q}$, $1 \leq i < j \leq m$, we say that n_i is the *initial node* of $\bar{Q}'$ and n_{j+1} is the *final node* of $\bar{Q}'$.

3 Defining checking sequences

Let us suppose that $\bar{D}$ is a distinguishing sequence for FSM M with n states, we apply an input sequence to the IUT M_I, and for every state s_i of M the resulting input/output sequence contains the subsequence $\bar{D}/\lambda(s_i, \bar{D})$. Then, since $M_I \in \Phi_M$ and so has at most n states, $\bar{D}$ must also be a distinguishing sequence for M_I. Further, $\bar{D}$ defines a bijection between the states of M and M_I. This motivates the following definitions, based on those in [25], of what it means to recognise a node in the label $\bar{Q}$ of a walk $\bar{P}$ and to verify a transition in $\bar{Q}$.

Definition 2 Let us suppose that $\bar{P}$ is a walk of $G(M)$ with starting vertex v_1 and label $\bar{Q}$.

1. A node n_i of $\mathit{Linear}(\bar{Q})$ is *d-recognised* in $\bar{Q}$ *as state* s of M if $\bar{D}/\lambda(s, \bar{D})$ is the label of a walk of $\mathit{Linear}(\bar{Q})$ with starting node n_i. This is illustrated in Figure 2 part 1).
2. A node n_i of $\mathit{Linear}(\bar{Q})$ is *d-recognised* in $\bar{Q}$ *as state* s_1 of M if $r/-$ is the label of a walk of $\mathit{Linear}(\bar{Q})$ that ends at node n_i. This is illustrated in Figure 2 part 2).
3. Let us suppose that $(n_q, n_i, \bar{T})$ and $(n_j, n_k, \bar{T})$ are walks of $\mathit{Linear}(\bar{Q})$ and $\bar{D}/\lambda(s, \bar{D})$ is a prefix of $\bar{T}$ (and thus n_q and n_j are d-recognised in $\bar{Q}$ as state s). Suppose also that node n_k is d-recognised as state s' of M. Then n_i is *t-recognised* in $\bar{Q}$ as s'. This is illustrated in Figure 2 part 3).
4. Let us suppose that $(n_q, n_i, \bar{T})$ and $(n_j, n_k, \bar{T})$ are walks of $\mathit{Linear}(\bar{Q})$ such that n_q and n_j are either d-recognised or t-recognised in $\bar{Q}$ as state s and n_k is either d-recognised or t-recognised in $\bar{Q}$ as state s'. Then n_i is *t-recognised* in $\bar{Q}$ as s'. This is illustrated in Figure 2 part 4).

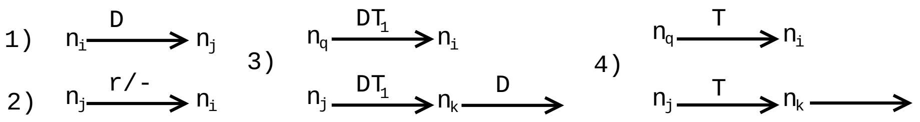
1) $n_i \xrightarrow{D} n_j$
2) $n_j \xrightarrow{r/-} n_i$
3) $n_q \xrightarrow{DT_1} n_i$
$n_j \xrightarrow{DT_1} n_k \xrightarrow{D}$
4) $n_q \xrightarrow{T} n_i$
$n_j \xrightarrow{T} n_k \longrightarrow$

Fig. 2 Recognising nodes

5. If node n_i of $Linear(\bar{Q})$ is either d-recognised or t-recognised in $\bar{Q}$ as state s then n_i is *recognised* in $\bar{Q}$ as state s.
6. Transition $\tau = (s_a, s_b, x/y)$ is *verified* in $\bar{Q}$ if there is an edge $(n_i, n_{i+1}, x_i/y_i)$ of $Linear(\bar{Q})$ such that n_i is recognised in $\bar{Q}$ as s_a, n_{i+1} is recognised in $\bar{Q}$ as s_b, $x_i = x$ and $y_i = y$.

The difference between this and the definition in [25] is the inclusion of the rule that the node following a reset is recognised as s_1. This rule has been added in order to reflect the reset being a reliable reset. Note that the distinguishing sequence $\bar{D}$ is an implicit parameter of this definition. These terms can be used to define a sufficient condition for an input sequence to be a checking sequence. The following result is based on Theorem 1 from [25].

Theorem 1 *Let $\bar{P}$ be a walk of $G(M)$ that starts at v_1, $\bar{Q} = label(\bar{P})$, and let us suppose that the initial node of $Linear(\bar{Q})$ is d-recognised as state s_1 in $\bar{Q}$. If every transition of M is verified in $\bar{Q}$ then the input portion of $\bar{Q}$ is a checking sequence of M.*

There is only one small difference between this result and Theorem 1 from [25]. This is that [25] gives a different definition of a checking sequence in that a checking sequence is required to distinguish between M and any element of Φ_M that is not isomorphic to M. As a result, a checking sequence under the definition of [25] need not detect the IUT starting in the wrong state. Since we require that the IUT and M have equivalent initial states we add the condition that the initial node of $Linear(\bar{Q})$ is d-recognised as state s_1.

In checking sequence generation we recognise the ending vertex of an edge of $G(M)$ that represents transition τ through the use of a distinguishing sequence $\bar{D}$; the corresponding subsequence included in a walk $\bar{P}$ is called a *test subsequence* for τ.

4 Generating a checking sequence

This section gives an algorithm for generating a checking sequence from M on the basis of a distinguishing sequence $\bar{D}$. It starts by defining α'-sequences [18]. We then adapt the algorithm of [18] in order to minimise the number of reset transitions in the resultant checking sequence.

4.1 Defining α'-sequences

In previous work [18] input/output sequences called α'-sequences were used as the basis for generating a checking sequence. The set of α'-sequences used is called an α'-set. For each state s_i there is an α'-sequence that is the label of a walk of $G(M)$, includes a subsequence $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ for some input/output sequence $\bar{T}_i$, and contains this subsequence in a context in which its ending node of $Linear(\bar{\alpha}_i)$ is d-recognised. For each state s_i of M, $\bar{T}_i$ is the label of a walk of $G(M)$ that starts at the vertex corresponding to $\delta(s_i, \bar{D})$ and ends at a vertex of $G(M)$ corresponding to a state s_j of M. Each $\bar{T}_i$, $1 \leq i \leq n$, is called a *transfer sequence*. The proposed approach will be parameterised by the $\bar{T}_i$ and in practice we will produce the α'-sequences once the $\bar{T}_i$ have been defined. First we define α'-sequences and we then outline how they can be generated once the $\bar{T}_i$ have been defined, explaining the algorithm of [18].

Before defining α'-sequences we define a set of transfer sequences.

Definition 3 Given FSM M, a *transfer set* T is a set $\{\bar{T}_1, \ldots, \bar{T}_n\}$ of labels of walks of $G(M)$ such that for each state s_i of M the sequence $\bar{T}_i$ is the label of a walk of $G(M)$ whose starting vertex corresponds to the state $\delta(s_i, \bar{D})$. Each element of a transfer set is called a *transfer sequence*.

The α'-sequences can be defined in the following way.

Definition 4 Given a transfer set $T = \{\bar{T}_1, \ldots, \bar{T}_n\}$ a set A of input/output sequences that are labels of walks of $G(M)$ is *an α'-set* if it satisfies the following conditions.

1. For each element $\bar{\alpha}_i$ of A there exist some $i_1, \ldots, i_k$ such that $\bar{\alpha}_i = \bar{D}/\lambda(s_{i_1}, \bar{D})\bar{T}_{i_1} \ldots \bar{D}/\lambda(s_{i_k}, \bar{D})\bar{T}_{i_k}$.
2. For each state s_i of M, there is a sequence $\bar{\alpha}_k$ in A and state s_j of M such that $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i\bar{D}/\lambda(s_j, \bar{D})$ is a subsequence of $\bar{\alpha}_k$.
3. For each element $\bar{\alpha}_i$ of A there are states s_j, s_j' and an element $\bar{\alpha}_k$ of A such that $\bar{\alpha}_i$ has a suffix of the form $\bar{D}/\lambda(s_j, \bar{D})\bar{T}_j$ and $\bar{D}/\lambda(s_j, \bar{D})\bar{T}_j\bar{D}/\lambda(s_j', \bar{D})$ is a subsequence of $\bar{\alpha}_k$.

Each element of an α'-set is called an *α'-sequence*.

Definition 4 ensures that the α'-sequences have the following properties when included in an input/output sequence that is the input portion of the label $\bar{Q}$ of a walk $\bar{P}$ in $G(M)$ that starts at v_1 and contains each α'-sequence.

1. As a result of the first requirement in the definition: The input portion of an α'-sequence starts with $\bar{D}$ and so an α'-sequence can be used to check the ending state of a transition.
2. As a result of the second requirement in the definition: If the α'-sequences are labels of walks in M_I then $\bar{D}$ must be a distinguishing sequence in M_I since M_I has at most n states and n distinct responses to $\bar{D}$ are observed in the α'-sequences.
3. As a result of the second requirement in the definition: If there is a walk of $Linear(\bar{Q})$ from node n to node n' with label $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ then n' is recognised as the state s_j reached from s_i by a transition sequence with label $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ since n is d-recognised as s_i and the ending node of a walk with label $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ is d-recognised as s_j in an α'-sequence.
4. As a consequence the third requirement, the ending node of the walk for each α'-sequence is recognised.

4.2 Generating α'-sequences

In this section we briefly outline the algorithm given in [18] for generating the α'-set. This algorithm is parameterised by the $\bar{T}_i$, $1 \leq i \leq n$, and the application of the

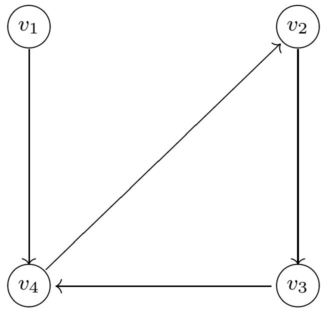

Fig. 3 The digraph $G_{\bar{D}}$

$\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ can be represented by a digraph $G_{\bar{D}} = (V, E_{\bar{D}})$ in which an edge from v_i represents a walk with label $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ and has ending vertex v_j such that the transition sequence of M with label $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ has ending state s_j. Each vertex of $G_{\bar{D}}$ has one edge leaving it.

To construct an α'-set A we can first produce a set $P = \{\bar{\rho}_1, \ldots, \bar{\rho}_q\}$ of paths and cycles of $G_{\bar{D}}$ such that every edge of $G_{\bar{D}}$ is included exactly once in an element of P. For each $\bar{\rho}_k \in P$, we then produce the input/output sequence $seq(\bar{\rho}_k) = label(\bar{\rho}_k)\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$, where v_i is the ending vertex of $\bar{\rho}_k$. This gives the α'-set $A = \{seq(\bar{\rho}_k) | \bar{\rho}_k \in P\}$. The problem of generating an α'-set can thus be reduced to that of producing such a set P of paths and cycles given $G_{\bar{D}}$ (and thus from the $\bar{T}_i$). A low order polynomial algorithm has been devised for producing such an α'-set that minimises its overall contribution to the checking sequence length [18][3]. We do not repeat this algorithm here.

Consider M_0, distinguishing sequence $\bar{D} = aa$, and $\bar{T}_i = \epsilon$ for all $1 \leq i \leq 4$. We have that $\delta(s_1, aa) = s_4$, $\delta(s_2, aa) = s_3$, $\delta(s_3, aa) = s_4$, and $\delta(s_4, aa) = s_2$. The digraph $G_{\bar{D}}$ produced for M_0, using empty transfer sequences, is given in Figure 3. We could choose any one of several sets of paths and cycles for P including the set that contains a path of length 1 from v_1 to v_4 and a cycle of length 3 from v_2 to v_2. This leads to the following α'-set:

1. sequence $\bar{\alpha}_1 = \bar{D}\bar{D}/0011$ from state s_1; and
2. sequence $\bar{\alpha}_2 = \bar{D}\bar{D}\bar{D}\bar{D}/01101101$ from state s_2.

We can see that this is an α'-set since in each case the final application of $\bar{D}$ is contained in the body of one of the α'-sequences: $\bar{\alpha}_1$ ends in $\bar{D}$ from s_4 and this is in the body of $\bar{\alpha}_2$; $\bar{\alpha}_2$ ends in $\bar{D}$ from s_2 and this is in the body of $\bar{\alpha}_2$. There are alternative choices such as one sequence $\bar{D}\bar{D}\bar{D}\bar{D}\bar{D}$, which labels a walk of $G(M)$ with starting vertex v_1.

[3] We do not require an α'-set that has shortest overall length. This is because in checking sequence generation, each α'-sequence will replace one $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ in checking the ending state of a transition.

4.3 A sufficient condition

We now give a sufficient condition, from [18], for a test sequence to be a checking sequence. This defines a set of checking sequences, for an FSM M and α'-set A and in Subsection 4.4 we consider the problem of finding a checking sequence from this set with a minimum number of reset transitions. Note that later, in Theorem 4, we prove a more general result and so we do not include a proof of Theorem 2 here.

Theorem 2 *Let A denote an α'-set and let us suppose that edge set $E_{\mathcal{T}}$, that represents transition sequences of M, has the following properties.*

1. *For each (non-reset) transition τ, with ending state s_j, $E_{\mathcal{T}}$ contains one edge representing τ followed by either a walk with label $\bar{D}/\lambda(s_j, \bar{D})\bar{T}_j$ or a walk with label $\bar{\alpha}_k$ for an α'-sequence $\bar{\alpha}_k \in A$ with prefix $\bar{D}/\lambda(s_j, \bar{D})\bar{T}_j$.*
2. *For every α'-sequence $\bar{\alpha}_k$ from A, $E_{\mathcal{T}}$ contains one edge that represents either a walk with label $\bar{\alpha}_k$ or a (non-reset) transition τ followed by a walk with label $\bar{\alpha}_k$.*
3. *Every edge from $E_{\mathcal{T}}$ represents either an α'-sequence or a (non-reset) transition τ, with ending state s_j, followed by either $\bar{D}/\lambda(s_j, \bar{D})\bar{T}_j$ or an α'-sequence $\bar{\alpha}_k \in A$ with prefix $\bar{D}/\lambda(s_j, \bar{D})\bar{T}_j$.*

Let Υ denote a tour of digraph $(V, E \cup E_R \cup E_{\mathcal{T}})$ that includes every edge from $E_{\mathcal{T}}$. Let e denote an edge from Υ that has starting vertex v_i of $G(M)$ reached from v_1 by a walk with label $\bar{D}/\lambda(s_1, \bar{D})\bar{T}_1$. Let $\bar{P}$ denote the walk produced by starting Υ with e. Let us suppose that E_{con} is the set of edges in Υ that are not in $E_R \cup E_{\mathcal{T}}$ and $G(M)[E_{con}]$ is acyclic. Then the input portion of $\bar{Q} = \bar{D}/\lambda(s_1, \bar{D})\bar{T}_1 label(\bar{P})$ is a checking sequence for M.

Throughout the paper, when we generate a tour Υ with a required set of edges E' we use E_{con} to denote the set of edges that are in Υ but not in E'.

Given M, distinguishing sequence $\bar{D}$ and α'-set A, the set $E_{\mathcal{T}}$ is not uniquely defined. Thus, checking sequence generation can be seen in terms of choosing some $E_{\mathcal{T}}$ and generating a checking sequence from this. However, the two parts of this process can be combined into one optimisation algorithm that chooses the optimal $E_{\mathcal{T}}$ and a corresponding optimal checking sequence [18].

The α'-sequences are defined in terms of a set of transfer sequences $\bar{T}_1, \ldots, \bar{T}_n$. The algorithm given in this paper can thus be seen as being parameterised by this set of transfer sequences. Section 5 reports the results of experiments that explore a heuristic: using empty transfer sequences in the α'-sequences. The intuition behind this heuristic is that using empty $\bar{T}_i$ allows greater freedom of choice regarding the transitions that follow the verification of a transition and this might assist in limiting the number of resets used. Note that this heuristic was used in the case where we simply wish to produce a shortest checking sequence and do not consider the number of resets included [18].

4.4 An optimisation algorithm

This section gives an algorithm that represents checking sequence generation as an optimisation problem. The first step is to represent the problem as an instance of the RCPP for a new digraph $TestG(M)$ that is produced such that a minimum cost tour, that contains the required edges, defines an optimal checking sequence. We then produce a minimum cost symmetric augmentation of the set of required edges. If the resultant digraph $Aug(M)$ is strongly connected then it has an Euler tour and we form the checking sequence from this Euler Tour. If $Aug(M)$ is not strongly connected then we need to add walks from $G(M)$ in order to connect its components. In Subsection 4.5 we show how such walks, that contain no reset transitions, can be generated. We now describe the optimisation algorithm used.

Recall that M without reset transitions is represented by $G(M) = (V, E)$ and the edge set E_R represents the reset transitions. We want to produce a walk $\bar{P}$ of digraph $(V, E \cup E_R)$ that satisfies the conditions of Theorem 2. We can consider the problem as being one of connecting a set of subsequences where each subsequence is either an α'-sequence or is a (non-reset) transition $\tau = (s_i, s_j, x/y)$ followed by either a walk with label $\bar{D}/\lambda(s_j, \bar{D})\bar{T}_j$ or an α'-sequence. Further, we require that the set of additional connecting transitions defines an acyclic digraph.

In a similar way to [16] we define an upper bound on the length of the checking sequence; this will be used to punish reset transitions in the checking sequence generation algorithm. By Theorem 2, at worst the checking sequence is a set of test subsequences and α'-sequences connected to form one sequence. The sum of the lengths of the subsequences connected to form a checking sequence is bounded above by the sum of the lengths of the subsequences formed by following each transition τ (with ending state s_i) by a walk with label $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ and the sum of the lengths of the α'-sequences. Let $T_m = max\{|\bar{T}_i|, 1 \leq i \leq n\}$. Since M has $|X||S|$ transitions, this gives an upper bound of $|X||S|(1 + |\bar{D}| + T_m) + \sum_{i=1}^{k} |\bar{\alpha}_i|$ on the overall length of the subsequences to be connected. In forming a tour from these $|X||S| + |A|$ subsequences, there are connecting walks between any two subsequences and each of these has length at most $|S| - 1$. Thus, the overall checking sequence length is bounded above by $U = |X||S|(1 + |\bar{D}| + T_m) + \sum_{i=1}^{k} |\bar{\alpha}_i| + (|X||S| + |A|)(|S| - 1)$. In the example M_0, we have $T_m = 0$, $|X| = 2$, $|S| = 4$, $|\bar{D}| = 2$, $|A| = 2$, and $\sum_{i=1}^{k} |\bar{\alpha}_i| = 12$ and so we use $U = 2 \times 4 \times (1 + 2 + 0) + 12 + (2 \times 4 + 2) \times 3 = 66$.

Given walk $\bar{P}$ of $G(M)$ we can associate a cost with $\bar{P}$. We give each edge in E cost 1 and each edge in E_R cost U. Since U is an upper bound on the overall checking sequence length, a minimum cost tour is also a tour with a minimum number of reset transitions (Proposition 3 below). We include a cost for each edge in E since ideally we would like to produce a shortest checking sequence amongst those that minimise the number of resets. We now give an algorithm for producing a minimum cost tour. This algorithm represents the problem in terms of the RCPP in a digraph $TestG(M)$. Algorithm 1 shows how $TestG(M) = (V', E')$ can be produced, where $E' = E \cup E_t \cup E_{\bar{D}} \cup E_\alpha \cup E_\epsilon \cup E_R$ for sets of edges defined in the algorithm.

The digraph $(V', E_t \cup E_{\bar{D}} \cup E_\alpha)$ produced for M_0 with distinguishing sequence $\bar{D} = aa$ and α'-set $\{\bar{\alpha}_1, \bar{\alpha}_2\}$ is shown in Figure 4. Here only the edges from $E_t \cup E_{\bar{D}} \cup E_\alpha$ in $TestG(M_0)$ are given. Thus, the lines from vertices of the form v_i represent edges of $G(M_0)$ and there are eight such lines. The lines from vertices of the form v'_i represent the application of $\bar{D}$ and the two α'-sequences and so there are six such lines.

The following is the key property of $TestG(M)$ that corresponds to the requirements of Theorem 2.

Proposition 1 *Let us suppose that Υ is a tour of $TestG(M)$ that includes every edge from $E_t \cup E_\alpha$. Then label(Υ) is the label of a tour of $G(M)$ with subwalks from a set P_T that satisfies the following conditions.*

Algorithm 1 Generating $TestG(M)$

```
Input $M = (S, X, Y, \delta, \lambda, s_1)$, $\bar{D}$, $\bar{T}_1, \ldots, \bar{T}_n$, and $\alpha$-set $A = \{\bar{\alpha}_1, \ldots, \bar{\alpha}_q\}$
Let $V' = V \cup \{v'_i | v_i \in V\}$.
    Comment: We have two copies of each state $s_i$: vertices $v_i$ and $v'_i$. Here vertex $v'_i$ represents the situation in which we have reached $s_i$ through a transition that is part of a transition test and thus whose ending state must be checked (by either $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ or an $\alpha'$-sequence $\bar{\alpha}_k \in A$ with prefix $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$).
Let $E = \{(v_i, v_j, x/y) | s_i, s_j \in S \wedge x \in X \wedge \delta(s_i, x) = s_j \wedge \lambda(s_i, x) = y\}$.
    Comment: The edges from $E$ represent the transitions of $M$ and allow us to connect the transition tests.
Let $E_t = \{(v_i, v'_j, x/y) | s_i, s_j \in S \wedge x \in X \wedge \delta(s_i, x) = s_j \wedge \lambda(s_i, x) = y\}$.
    Comment: Since each transition is to be tested by having its ending state checked, for each transition $(s_i, s_j, x/y)$ there is a corresponding edge $(v_i, v'_j, x/y)$.
Let $E_{\bar{D}}$ be the set of edges of the form $(v'_i, v_j, \bar{D}/\lambda(s_i, \bar{D})\bar{T}_i)$ for $v'_i$, where $v_j$ is the ending vertex of the walk in $G(M)$ that has starting vertex $v_i$ and label $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$.
    Comment: These edges represent the use of the $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ to recognise the ending states of transitions with ending state $s_i$.
Let $E_\alpha$ denote the set of $(v'_i, v_j, \bar{\alpha}_k)$ such that $\bar{\alpha}_k \in A$ labels a walk of $G(M)$ with starting vertex $v_i$ and ending vertex $v_j$.
    Comment: These edges represent the use of $\alpha'$-sequences to recognise the ending states of transitions.
Let $E_\epsilon = \{(v_i, v'_i, \epsilon) | v_i \in V\}$.
    Comment: If we just use the other edges then we can only execute an $\alpha'$-sequence as part of a transition test. The edges in $E_\epsilon$ allow an $\alpha'$-sequence to be executed separately from the transition tests. Thus the inclusion of an edge from $E_\epsilon$ in a tour does not introduce additional input.
Let $E_R = \{(v_i, v_1, r/-) | 1 \leq i \leq n\}$.
    Comment: These edges represent reset transitions.
Output $TestG(M) = (V', E') = (V', E \cup E_t \cup E_{\bar{D}} \cup E_\alpha \cup E_\epsilon \cup E_R)$
```

1. *For each (non-reset) transition τ, with ending state s_j, $P_{\mathcal{T}}$ contains a walk representing τ followed by either a walk with label $\bar{D}/\lambda(s_j, \bar{D})\bar{T}_j$ or a walk with label $\bar{\alpha}_k$ for an α'-sequence $\bar{\alpha}_k \in A$ with prefix $\bar{D}/\lambda(s_j, \bar{D})\bar{T}_j$.*
2. *For every α'-sequence $\bar{\alpha}_k$ from A, $P_{\mathcal{T}}$ contains either a walk with label $\bar{\alpha}_k$ or the label of a (non-reset) transition τ followed by a walk with label $\bar{\alpha}_k$.*
3. *Every walk from $P_{\mathcal{T}}$ represents either an α'-sequence or a (non-reset) transition τ, with ending state s_j, followed by either an α'-sequence $\bar{\alpha}_k \in A$ with prefix $\bar{D}/\lambda(s_j, \bar{D})\bar{T}_j$ or $\bar{D}/\lambda(s_j, \bar{D})\bar{T}_j$.*

Proof : We choose a set $P_{\mathcal{T}}$ in the following way:

1. For every edge $e \in E_\alpha$ representing an α'-sequence choose a subwalk $\bar{w}$ of Υ of length two that has e as its second edge and include in $P_{\mathcal{T}}$ a walk of $G(M)$ with label $label(\bar{w})$. This is possible since we require that Υ contains every edge from E_α.
2. For each transition $\tau = (s_i, s_j, x/y)$ of M such that a walk representing τ followed by an α'-sequence $\bar{\alpha}_k \in A$ with prefix $\bar{D}/\lambda(s_j, \bar{D})\bar{T}_j$ or $\bar{D}/\lambda(s_j, \bar{D})\bar{T}_j$ has not been chosen, include in $P_{\mathcal{T}}$ a walk of $G(M)$ with a label that is the label of a subwalk of

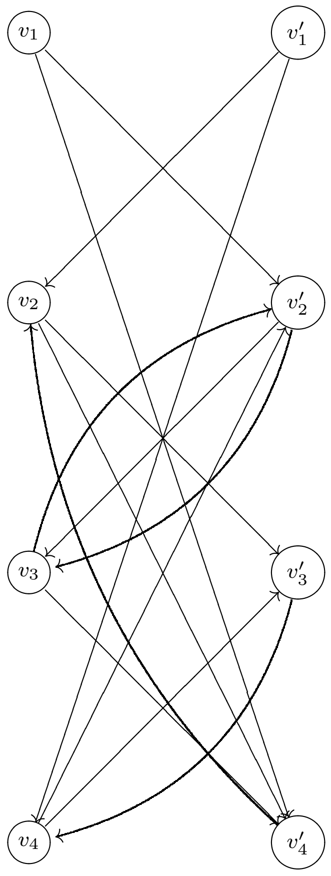

Fig. 4 The digraph $(V', E_t \cup E_{\bar{D}} \cup E_\alpha)$ for FSM M_0

Υ of length two whose first edge is $(v_i, v'_j, x/y)$. We can always choose some such walk since Υ is required to include every edge from E_t.

We now consider the three properties in the proposition.

The first property follows from the fact that a transition $\tau = (s_i, s_j, x/y)$ is represented by an edge in E_t from v_i to v'_j and in a tour this must be followed by an edge that either represents an α'-sequence $\bar{\alpha}_k \in A$ with prefix $\bar{D}/\lambda(s_j, \bar{D})\bar{T}_j$ or $\bar{D}/\lambda(s_j, \bar{D})\bar{T}_j$ since these are the only edges of $TestG(M)$ that can have starting vertex v'_j.

The second property is a consequence of the requirement that $P_{\mathcal{T}}$ contains every α'-sequence.

For the third property observe that the label of a walk in $P_{\mathcal{T}}$ is the label of a walk of $TestG(M)$ with length two whose first edge has ending vertex v'_j for some $1 \le j \le n$. In addition, the first edge of such a walk cannot represent a reset transition and can only represent an edge from E_ϵ if the second edge represents an α'-sequence. $\Box$

Each edge from $E \cup E_t$ is given cost 1, as each of these edges represents a single transition that is not a reset transition. Each edge from E_R is given cost U. An edge from E_ϵ is given cost 0, while edges from $E_{\bar{D}}$ and E_α are given a cost that represents the length of the corresponding input/output sequence[4]. We introduce some notation before proving that minimum cost tours minimise the number of resets.

Definition 5 Let $\varUpsilon(E_t, E_\alpha)$ denote the set of tours of $TestG(M)$ that include every edge from $E_t \cup E_\alpha$ exactly once.

We have the following property, which tells us that we lose nothing by considering only tours in $\varUpsilon(E_t, E_\alpha)$.

Proposition 2 *Every tour $\varUpsilon$ of $TestG(M)$ that includes every edge from $E_t \cup E_\alpha$ at least once has the same label as a tour of $TestG(M)$ that includes every edge from $E_t \cup E_\alpha$ exactly once.*

Proof : To see this let us first suppose that an edge $e \in E_t$ is repeated in a tour $\varUpsilon$ of $\varUpsilon(E_t, E_\alpha)$. Then we can take a subwalk of $\varUpsilon$ with length two that contains e and does not include an edge from E_α only included once in $\varUpsilon$. We can then replace this subwalk by a sequence of edges from E. We can repeat this process until the tour contains exactly one instance of each edge from E_t. Finally, if an edge from E_α with label $\bar{D}/\lambda(s_i, \bar{D})\bar{T}_i$ is repeated then we can replace all but one of the copies of this edge by the edge from $E_{\bar{D}}$ with starting vertex v'_i followed by a walk in (V, E) with label $\bar{T}$. □

Proposition 3 *Let us suppose that $\varUpsilon$ is a minimum cost element of $\varUpsilon(E_t, E_\alpha)$. Then $\varUpsilon$ is an element of $\varUpsilon(E_t, E_\alpha)$ with fewest reset transitions.*

Proof : Let $\varUpsilon'(E_t, E_\alpha)$ denote the set of tours of $TestG(M)$ in $\varUpsilon(E_t, E_\alpha)$ with the property that every cycle in a tour from $\varUpsilon'(E_t, E_\alpha)$ contains at least one edge from $E_t \cup E_\alpha$. Given a tour $\varUpsilon_1$ from $\varUpsilon(E_t, E_\alpha) \setminus \varUpsilon'(E_t, E_\alpha)$, we can delete at least one cycle from $\varUpsilon_1$ to produce a shorter tour $\varUpsilon_2$ from $\varUpsilon(E_t, E_\alpha)$ such that $\varUpsilon_2$ contains no more reset transitions than $\varUpsilon_1$. It is thus sufficient to only consider tours in $\varUpsilon'(E_t, E_\alpha)$.

Since $\varUpsilon$ is a minimum cost element of $\varUpsilon(E_t, E_\alpha)$ we have that $\varUpsilon$ is in $\varUpsilon'(E_t, E_\alpha)$. Further, it is a minimum cost member of $\varUpsilon'(E_t, E_\alpha)$. The result now follows from the cost of each edge representing a reset transition having cost U for a value U that is an upper bound on the length of the tours in $\varUpsilon'(E_t, E_\alpha)$. □

Checking sequence generation can be seen as the problem of finding a minimum cost element of $\varUpsilon(E_t, E_\alpha)$: this is an instance of the RCPP. Naturally, we must ensure that the set E_{con} of connecting transitions defines an acyclic digraph. We apply the following procedure, used in [1], for solving the RCPP[5]. First we find a minimum cost symmetric augmentation $Aug(M) = (V', E_{Aug})$ of the set $E_t \cup E_\alpha$ in $TestG(M)$ by adding copies of some edges from the set $E \cup E_{\bar{D}} \cup E_\epsilon \cup E_R$. This can be found in polynomial time [1]. If $Aug(M)$ is connected then it has an Euler Tour $\varUpsilon$ and this provides a solution to the RCPP.

Proposition 4 *Let us suppose that $Aug(M) = (V', E_{Aug})$ is the minimum cost symmetric augmentation of the set $E_t \cup E_\alpha$ in $TestG(M)$ and let E_{con} denote the set of edges in $E_{Aug} \setminus (E_t \cup E_{\bar{D}} \cup E_\alpha \cup E_R \cup E_\epsilon)$. Then the digraph $TestG(M)[E_{con}]$ is acyclic.*

[4] These edges represent sequences that do not include reset transitions.

[5] Since the RCPP is NP-hard, the polynomial time algorithm given in [1] does not always return the shortest test sequence. In contrast, we show that this algorithm can be adapted so that it is guaranteed to return a checking sequence with fewest reset transitions.

Proof : Proof by contradiction: let us suppose that $TestG(M)[E_{con}]$ contains cycles and let $e_1, \ldots, e_k$ $(k \geq 1)$ denote a minimum length cycle of $TestG(M)[E_{con}]$. Now consider the digraph $Aug'(M)$ produced from $Aug(M)$ by deleting one copy of each of $e_1, \ldots, e_k$. Then since $e_1, \ldots, e_k$ is a cycle and $Aug(M)$ is symmetric, $Aug'(M)$ is symmetric. Further, $Aug'(M)$ contains every edge from $E_t \cup E_\alpha$ and has lower cost than $Aug(M)$. This contradicts $Aug(M)$ being a minimum cost symmetric augmentation of $E_t \cup E_\alpha$ in $TestG(M)$, as required. □

If $Aug(M)$ is not strongly connected then it defines a set of components. In Subsection 4.5 we show how walks can be added in order to connect these components without adding reset transitions. If $Aug(M)$ is strongly connected then we produce a checking sequence in the following way. We choose an edge e in Υ that starts at the vertex v_i reached from v_1 by a walk with label $\bar{D}/\lambda(s_1, \bar{D})\bar{T}_1$. We start Υ with e to give walk $\bar{P}$ and return the input portion of $\bar{D}/\lambda(s_1, \bar{D})\bar{T}_1 label(\bar{P})$ as the checking sequence. This is summarised in Algorithm 2.

Algorithm 2 Checking sequence generation algorithm if $Aug(M)$ is connected

```
Calculate $U = |X||S|(1 + |\bar{D}| + T_m) + \sum_{i=1}^{k} |\bar{\alpha}_i| + (|X||S| + |A|)(|S| - 1)$, where $A = \{\bar{\alpha}_1, \ldots, \bar{\alpha}_k\}$ is the $\alpha'$-set used, $T_m$ is an upper bound on the lengths of the transfer sequences used, $X$ is the input alphabet of $M$ and $S$ is the state set of $M$.
Define the digraph $TestG(M)$ and find a minimum cost symmetric augmentation $Aug(M)$ of $E_t \cup E_\alpha$ in $TestG(M)$.
Find an Euler Tour $\Upsilon$ of $Aug(M)$.
Let $e$ denote an edge from $\Upsilon$ that has starting vertex $v_i$ reached from $v_1$ by a walk with label $\bar{D}/\lambda(s_1, \bar{D})\bar{T}_1$. Let $\bar{P}$ denote the walk produced by starting $\Upsilon$ with $e$.
Return the input portion of $\bar{Q} = \bar{D}/\lambda(s_1, \bar{D})\bar{T}_1 label(\bar{P})$.
```

Theorem 3 *Let us suppose that when Algorithm 2 is applied the digraph* $Aug(M)$ *is strongly connected. Then the resultant input portion of* $\bar{Q} = \bar{D}/\lambda(s_1, \bar{D})\bar{T}_1 label(\bar{P})$ *is a checking sequence that has a minimal number of reset transitions amongst those that satisfy the conditions of Theorem 2.*

Proof : From Theorem 2, Proposition 1 and Proposition 4 we know that $\bar{Q}$ is a checking sequence. From Proposition 3 we know that it minimises the number of reset transitions. □

Consider the FSM M_0. Here, solving the RCPP for the digraph $TestG(M)$ and the set of required edges leads to a strongly connected digraph that has the following Euler Tour.

$$v_1' \xrightarrow{\bar{\alpha}_1} v_2 \xrightarrow{t_{2b}} v_3' \xrightarrow{\bar{D}} v_4 \xrightarrow{t_{4a}} v_3' \xrightarrow{\bar{D}} v_4 \xrightarrow{t_{4b}} v_2' \xrightarrow{\bar{\alpha}_2} v_3 \xrightarrow{t_{3b}} v_4' \xrightarrow{\bar{D}} v_2 \xrightarrow{t_{2a}} v_4' \xrightarrow{\bar{D}} v_2$$

$$\xrightarrow{r/-} v_1 \xrightarrow{t_{1a}} v_2' \xrightarrow{\bar{D}} v_3 \xrightarrow{t_{3a}} v_2' \xrightarrow{\bar{D}} v_3 \xrightarrow{r/-} v_1 \xrightarrow{t_{1b}} v_4' \xrightarrow{\bar{D}} v_2 \xrightarrow{r/-} v_1 \xrightarrow{\epsilon} v_1'$$

We can thus obtain the following checking sequence by starting the tour at v_4 after the application of $\bar{D}$ since $\delta(s_1, \bar{D}) = s_4$. If we choose the first instance of v_4 above we get a checking sequence that contains three resets that is defined by:

$$\bar{D}a\bar{D}baaaaaaaa(= \bar{\alpha}_2)b\bar{D}a\bar{D}ra\bar{D}a\bar{D}rb\bar{D}raaaa(= \bar{\alpha}_1)b\bar{D}$$

This leads to the following checking sequence $aaaaabaaaaaaaabaaaaaraaaaaarbaaraaaabaa$. In this case the tour contains the vertex v_1'. Where this is the case, we have an alternative way of creating a checking sequence: we can start the tour at v_1' and add an instance of $\bar{D}$ to the end of the resultant sequence. In this case the final reset (and additional $\bar{D}$) can be eliminated giving a checking sequence with two resets.

$$aaaa(=\bar{\alpha}_1)b\bar{D}a\bar{D}baaaaaaaa(=\bar{\alpha}_2)b\bar{D}a\bar{D}ra\bar{D}a\bar{D}rb\bar{D}$$

There are two reasons why we can reduce the number of resets by one in this example. First, an α'-sequence starts at state s_1 and so we can start the checking sequence with this α'-sequence. Second, no transition ends at state s_1 and so in the tour the α'-sequence is not used in order to check the final state of a transition and it is preceded by a reset transition that can be eliminated. We require two resets since we cannot return to state s_1 once we have left it and the method requires us to have three edges that start at s_1: one for each transition with starting state s_1 and one for the α'-sequence α_1. This concludes our analysis of the case in which $Aug(M)$ is strongly connected.

4.5 Connecting components

This subsection considers the case where $Aug(M)$ is not strongly connected. We show how walks from $TestG(M)$ can be added to $Aug(M)$ in order to produce a strongly connected digraph $Aug'(M)$ such that the walks added contain no reset transitions and a checking sequence can be produced from $Aug'(M)$. We could adapt the results in [16] to show that we can add walks to $Aug(M)$ to make it connected without using reset transitions. However, such walks might introduce cycles into the set E_{con} of connecting edges and this is not allowed under Theorem 2.

The following weakening, of the condition for a test sequence to be a checking sequence, helps us to overcome this issue. This allows us to add walks, without including them in the set E_{con} of connecting edges, if each walk $\bar{P}$ added satisfies the following condition: the label of $\bar{P}$ ends in a subsequence of the form $\bar{D}/\lambda(s_i,\bar{D})\bar{T}_i$ for some $s_i \in S$. The reason we can add such a walk is that its final node is t-recognised as the corresponding state of M.

Theorem 4 *Let A denote an α'-set and let us suppose that the sets $E_{\mathcal{T}}$ and $E_{\mathcal{C}}$ of edges that correspond to transition sequences of M have the following properties.*

1. *For each (non-reset) transition $\tau = (s_i, s_j, x/y)$ of M, the set $E_{\mathcal{T}}$ contains one edge representing τ followed by either a walk with label $\bar{D}/\lambda(s_j,\bar{D})\bar{T}_j$ or a walk with label $\bar{\alpha}_k$ for an α'-sequence $\bar{\alpha}_k \in A$ with prefix $\bar{D}/\lambda(s_j,\bar{D})\bar{T}_j$.*
2. *For every α'-sequence $\bar{\alpha}_k$ from A, $E_{\mathcal{T}}$ contains one edge that represents either a walk with label $\bar{\alpha}_k$ or a (non-reset) transition τ followed by a walk with label $\bar{\alpha}_k$.*
3. *Every edge from $E_{\mathcal{T}}$ represents either an α'-sequence or a (non-reset) transition τ, with ending state s_j, followed by either an α'-sequence $\bar{\alpha}_k \in A$ with prefix $\bar{D}/\lambda(s_j,\bar{D})\bar{T}_j$ or $\bar{D}/\lambda(s_j,\bar{D})\bar{T}_j$.*
4. *For every edge e_j in $E_{\mathcal{C}}$ there exists a vertex $v_i \in V$ such that e_j represents a walk whose label has suffix $\bar{D}/\lambda(s_i,\bar{D})\bar{T}_i$.*

Let Υ denote a tour of digraph $(V', E \cup E_R \cup E_{\mathcal{T}} \cup E_{\mathcal{C}})$ that includes every edge from $E_{\mathcal{T}}$. Let e denote an edge from Υ that has starting vertex v_i reached from v_1 by a walk with label $\bar{D}/\lambda(s_1, \bar{D})\bar{T}_1$. Let $\bar{P}$ denote the walk produced by starting Υ with e. Let us suppose that E_{con} is the set of edges in Υ that are not in $E_{\mathcal{T}} \cup E_{\mathcal{C}} \cup E_R$ and $G(M)[E_{con}]$ is acyclic. Then the input portion of $\bar{Q} = \bar{D}/\lambda(s_1, \bar{D})\bar{T}_1 label(\bar{P})$ is a checking sequence for M.

Proof : From Theorem 1, it is sufficient to prove that each transition of M is verified in $\bar{Q}$. Since E_{con} is acyclic it is possible to place a partial ordering $\propto$ on V such that $v_i \propto v_j$ if and only if there is a path in (V, E_{con}) from v_i to v_j. This partial ordering can be extended to the nodes of $Linear(\bar{Q})$, which are ordered according to their corresponding vertices.

A proof by contradiction will be produced: assume that the input portion of $\bar{Q}$ does not represent a checking sequence. Then, by Theorem 1, some of the nodes of $Linear(\bar{Q})$ are not recognised. By definition, any node that is not recognised must follow an edge from E_{con}.

Amongst the nodes that are not recognised, take some n_i that corresponds to a vertex v_j that is minimal according to $\propto$. Here node n_i corresponds to vertex v_j of $G(M)$ if the prefix of $\bar{Q}$ of length i is a walk of M with ending state s_j. There may be more than one such minimal node, but any one will suffice.

It is now sufficient to look at the node n_{i-1} that precedes n_i (i cannot be 1, as the initial node is d-recognised as s_1 by $\bar{D}/\lambda(s_1, \bar{D})\bar{T}_1$). The edge from n_{i-1} to n_i must represent some edge $e \in E_{con}$, as its final node is not recognised, and thus $n_{i-1} \propto n_i$. By the minimality of n_i, n_{i-1} is recognised.

The edge e represents a transition τ of M. $Linear(\bar{Q})$ contains a subsequence, from node n_j say, that represents a test subsequence for τ. As $n_j \propto n_i$, by the minimality of n_i the node n_j must be recognised in $\bar{Q}$. Thus, in e', the transition τ exists within a context in which it is followed by $\bar{D}/\lambda(s, \bar{D})$ for some state s (possibly as part of an α'-sequence) and its initial node is recognised. Thus, by the definition of a node being recognised, as n_{i-1} is recognised n_i is also recognised. This provides a contradiction as required. □

We now prove a number of results regarding $Aug(M)$ that form the basis of the algorithm for adding walks to connect the components of $Aug(M)$.

Proposition 5 *If $Aug(M)$ is not strongly connected then it can be partitioned into a set of components.*

Proof : This follows from the fact that $Aug(M)$ is symmetric and any weakly connected symmetric subgraph is strongly connected. □

Thus the edge set E_{Aug} of $Aug(M)$ can be partitioned into maximal sets $C_1, \ldots, C_m$ such that each $Aug(M)[C_i]$ is strongly connected. We assume that such a partition exists and that v_1 is a vertex of the component $Aug(M)[C_1]$. We use the notion of the closure of a set of edges defined in [16].

Definition 6 Let us suppose that $C \subseteq E_{Aug}$ and $Aug(M)[C]$ is strongly connected. The *closure*, $cl(C)$, of C in $Aug(M)$ is the largest subset of $(E' \setminus E_R) \cup C$ such that $C \subseteq cl(C)$ and $TestG(M)[cl(C)]$ is strongly connected.

If C_i is the edge set of a component G_i of $Aug(M)$ then $cl(C_i)$ contains an edge with starting vertex v_1.

Theorem 5 *Let us suppose that $Aug(M)$ is the minimum cost symmetric augmentation of set $E_t \cup E_\alpha$ in $TestG(M)$ and $Aug(M)$ has components represented by edge sets $C_1, \ldots, C_m$. Then for all $2 \leq i \leq m$, the closure $cl(C_i)$ of C_i contains an edge with starting vertex v_1.*

Proof : Proof by contradiction: assume that for some $2 \leq i \leq m$, $cl(C_i)$ does not have an edge with starting vertex v_1. Note that each C_i must have an edge with a starting vertex in V and an edge with an ending vertex in V. There are two cases to consider.

Case 1: There does not exist an edge of $E \setminus cl(C_i)$ that has a starting vertex in $TestG(M)[cl(C_i)]$. Since $cl(C_i)$ does not contain an edge with starting vertex v_1, some edge $e = (v_l, v_j, x/y) \in E \setminus cl(C_i)$ (some $v_l, v_j \in V, x \in X, y \in Y$) has ending vertex in $TestG(M)[cl(C_i)]$. Consider the corresponding edge $e' = (v_l, v_j', x/y)$ from E_t and some edge e'' in E_{Aug} that has starting vertex v_j'. Since $TestG(M)$ is symmetric there must be some such e'' and e'' must represent either an α-sequence or a sequence whose input portion starts with $\bar{D}$. Recall that e'' represents a sequence of edges from E and thus contains no reset transitions and so since no edge of $E \setminus cl(C_i)$ has a starting vertex from $TestG(M)[cl(C_i)]$, the ending vertex of e'' must be in $TestG(M)[cl(C_i)]$. Further, e' and e'' must be in the same component of $Aug(M)$ since the ending vertex of e' is the starting vertex of e''. Thus since $e \in E \setminus cl(C_i)$, $TestG(M)[cl(C_i)]$ is strongly connected and v_l is not a vertex of $TestG(M)[cl(C_i)]$, e' and e'' must be in a component C_j such that $cl(C_j) \neq cl(C_i)$. Thus, $cl(C_i)$ and $cl(C_j)$ have edges connected to the ending vertex of e'' and so, since $TestG(M)[cl(C_i)]$ and $TestG(M)[cl(C_j)]$ are strongly connected, $TestG(M)[cl(C_i) \cup cl(C_j)]$ is strongly connected. By the maximality of $cl(C_i)$ and $cl(C_j)$, $cl(C_i) = cl(C_j)$. This provides a contradiction as required.

Case 2: There exists an edge $e = (v_l, v_j, x/y) \in E \setminus cl(C_i)$ that has a starting vertex in $TestG(M)[cl(C_i)]$. Consider the corresponding edge $e' = (v_l, v_j', x/y) \in E_t$ and some edge e'' in E_{Aug} that has starting vertex v_j'. Since e'' represents a sequence that contains no reset transitions, $TestG(M)[cl(C_i)]$ is strongly connected, and $e \notin cl(C_i)$, the ending vertex of e'' cannot be in $TestG(M)[cl(C_i)]$. Since e' and e'' are in the same component of $TestG(M)$ they are in some C_j such that $cl(C_j) \neq cl(C_i)$. Since $cl(C_i)$ and $cl(C_j)$ both have edges connected to the starting vertex of e', $cl(C_i) = cl(C_j)$. This provides a contradiction as required. $\square$

In [16] a similar result is used to show that for each component $Aug(M)[C_i]$ ($2 \leq i \leq m$) we can add a cycle of edges from E that connects $Aug(M)[C_i]$ to v_1 such that the cycle contains no reset transitions (recall that the reset transitions are not represented by edges from E). However, in producing a checking sequence as opposed to a test sequence we require more: we need to ensure that the walks we add lead to a tour that satisfies the conditions of Theorem 4 and thus lead to a checking sequence.

We will introduce an iterative algorithm, Algorithm 3, that adds edges to $Aug(M)$ in order to create a strongly connected symmetric digraph $Aug'(M)$. This is based on the following consequence of Theorem 5.

Proposition 6 *Let us suppose that $Aug'(M)$ has been formed from $Aug(M)$ by adding zero or more cycles formed by edges of E and $Aug'(M)$ is not strongly connected. If C is the edge set of the component of $Aug'(M)$ that contains v_1 then there is another edge set C_a of a component of $Aug'(M)$ such that there is an edge e in E from a vertex from $Aug'(M)[C_a]$ to a vertex of $Aug'(M)[C]$.*

Proof : Given $C_i \neq C$, the closure of C_i contains an edge with starting vertex v_1 and is strongly connected. Thus, there must be a walk in $TestG(M)$, that contains no

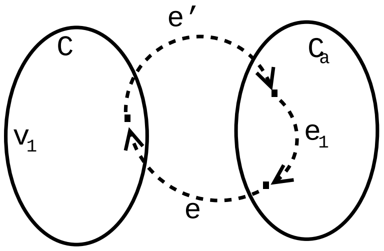

Fig. 5 Connecting components

reset transitions, from a vertex in $Aug'(M)[C_i]$ to v_1. In addition, $Aug'(M)[C_i]$ must contain vertices from V and since $Aug'(M)[C_i]$ is strongly connected we can choose a walk that starts at a vertex from V. Now observe that this walk starts at a vertex v_i in V, ends in a vertex v_1 in V, and contains no resets and so there must also be a walk in $TestG(M)$ from v_i to v_1 that contains only edges from E. Since all vertices of $Aug'(M)$ are either starting or ending vertices of edges of the C_j, this path must include an edge e from a vertex in $Aug'(M)[C_a]$ to a vertex in $Aug'(M)[C]$ for some $C_a \neq C$ and so the result follows. □

We can therefore choose such an edge $e = (v_i, v_j, x/y)$ from a component C_a of $Aug(M)$ to the component C_1. $Aug(M)$ also contains a walk that represents the testing of the transition corresponding to e by following edge $(v_i, v_j', x/y)$ by an edge from v_j' to some v_k whose label $\bar{T}$ is either a $\bar{D}/\lambda(s_i\bar{D})\bar{T}_i$ or an α'-sequence with prefix $\bar{D}/\lambda(s_i\bar{D})\bar{T}_i$. Since $Aug(M)[C_a]$ is strongly connected, v_k is a vertex in $Aug(M)[C_a]$. We can therefore define an edge e' from v_j to v_k with label $\bar{T}$ that corresponds to a walk in $TestG(M)$. Since $Aug(M)[C_a]$ is strongly connected there is a path $\bar{P}_1$ in $Aug(M)[C_a]$ from the ending vertex v_k of e' to the starting vertex v_i of e. If we add e, e', and an edge e_1 representing $\bar{P}_1$ to $Aug(M)$ then we have connected C and C_a and the new digraph is symmetric since the edges added form a cycle. This is illustrated in Figure 5. The edge e' can be included in a tour without having to add it to the set E_{con} of connecting edges in the conditions of Theorem 4, since it ends in a subsequence of the form $\bar{D}/\lambda(s_i\bar{D})\bar{T}_i$. We do not have to add any of the edges in $\bar{P}_1$ to E_{con} since these are already in $Aug(M)$. Below, in Proposition 7 we prove that the addition of e to E_{con} cannot introduce cycles.

At each step of Algorithm 3 some edges e, e' and e_1 are added to connect some C_a to C. Edge e' represents a path that, according to Theorem 4, can be added without including it in E_{con}. Edge e_1 represents a sequence of edges already included in C_a and thus it can be added without being added to E_{con}. Thus e is the only edge that we have to add to E_{con} in an iteration. The edges added in an iteration connect the components $Aug(M)[C]$ and $Aug(M)[C_a]$, do not include edges from E_R, and preserve the property of the digraph being symmetric.

Proposition 7 *If Algorithm 3 is applied to digraph $Aug(M)$ that is not strongly connected then the edge set E'' returned has the property that (V, E'') is acyclic.*

Proof : Each iteration of the algorithm involves adding an edge e to E'' such that e goes from a vertex of some $Aug(M)[C_a]$ to a vertex of the current $Aug(M)[C]$. Let E_I'' denote the set E'' before the algorithm is applied, let E_F'' denote the set E'' after the algorithm is applied, and let $E_F'' \setminus E_I'' = \{e_1, \ldots, e_m\}$ where the ith iteration of the algorithm adds the edge e_i to E', $1 \leq i \leq m$.

Algorithm 3 Connecting the components

```
Input $E'' = E_{con}$, $C = C_1$, $C' = \{C_2, \dots, C_m\}$.
**while** $C' \neq \emptyset$ **do**
    Choose an edge $e = (v_i, v_j, l) \in E$ such that $v_i$ is in $Aug(M)[C_a]$ for some $C_a \in C'$ and $v_j$ is in $Aug(M)[C]$.
        *Comment: By Proposition 6, there must be some such $e$ and $C_a$.*
    Find the edge $e'' = (v'_j, v_k, \bar{T})$ of $Aug(M)$ that represents a sequence that can be used to check the final state of the transition corresponding to $e$. Since $Aug(M)$ is strongly connected there must be some such $e''$. In addition, $\bar{T}$ is either $\bar{D}/\lambda(s_j\bar{D})\bar{T}_j$ or an $\alpha'$-sequence.
    Let $e' = (v_j, v_k, \bar{T})$.
    Produce a walk $\bar{P}_1$ from the ending vertex $v_k$ of $e'$ to the starting vertex $v_i$ of $e$ using edges from $C_a$ only and represent $\bar{P}_1$ by an edge $e_1$.
        *Comment: This is possible since $Aug(M)[C_a]$ is strongly connected. Further, we know that $C_a$ does not contain edges from $E_R$ since $v_1$ is not in $Aug(M)[C_a]$.*
    Let $C = C \cup C_a \cup \{e, e', e_1\}$, $C' = C' \setminus \{C_a\}$, and $E'' = E'' \cup \{e\}$.
**end while**
Output $C$ and $E''$.
```

Proof by contradiction: let us suppose that (V, E''_F) contains at least one cycle. First observe that, by Proposition 4, (V, E''_I) is acyclic. Let j be the integer such that $(V, E''_I \cup \{e_1, \dots e_{j-1}\})$ is acyclic and $(V, E''_I \cup \{e_1, \dots e_j\})$ contains cycles and let $\bar{P}$ be a minimum length cycle in $(V, E''_I \cup \{e_1, \dots e_j\})$. Let us suppose that the jth iteration involved adding an edge to connect C_a to C. Since $Aug(M)[C_a]$ and $Aug(M)[C]$ are strongly connected components that have no vertices in common and $E''_I \cup \{e_1, \dots e_j\}$ contains no edge from $Aug(M)[C]$ to $Aug(M)[C_a]$, $\bar{P}$ cannot contain e_j and thus must be a cycle in $(V, E''_I \cup \{e_1, \dots e_{j-1}\})$. This contradicts the minimality of j as required. □

Proposition 8 *Let us suppose that Algorithm 3 returns the sets C and E''. Then $Aug'(M) = (V', C \cup E'')$ is symmetric and strongly connected.*

Proof : We know that $Aug'(M)$ is strongly connected since in each iteration the edges added connect an element of C' to C. In addition, $Aug'(M)$ is symmetric since in each iteration we add a set of edges that forms a cycle. □

4.6 The overall checking sequence generation algorithm

We can now state the complete checking sequence algorithm, Algorithm 4.

The proof of the following is equivalent to that of Proposition 3

Proposition 9 *Let us suppose that Υ is a minimum cost tour of $Aug'(M)$ that contains every element of $E_t \cup E_\alpha$. Then amongst all tours of $Aug'(M)$ that contain every element of $E_t \cup E_\alpha$, Υ minimises the number of reset transitions.*

Theorem 6 *The input portion of $\bar{Q}$ produced by Algorithm 4 is a checking sequence that, amongst the checking sequences satisfying the conditions of Theorem 4, minimises the number of reset transitions used.*

Algorithm 4 The Overall Checking Sequence Algorithm

```
Calculate $U = |X||S|(1 + |\bar{D}| + T_m) + \sum_{i=1}^{k} |\bar{\alpha}_i| + (|X||S| + |A|)(|S| - 1)$.
Define the digraph $TestG(M)$ and find a minimum cost symmetric augmentation $Aug(M)$ of $E_t \cup E_\alpha$ in $TestG(M)$.
if $Aug(M)$ is not strongly connected then
    Apply Algorithm 3 to produce sets $C$ and $E''$ and form $Aug'(M) = (V', C \cup E'')$
else
    $Aug'(M) = Aug(M)$
end if
Find an Euler Tour $\Upsilon$ of $Aug'(M)$.
Let $e$ denote an edge from $\Upsilon$ that has starting vertex $v_i$ reached from $v_1$ by a walk with label $\bar{D}/\lambda(s_1, \bar{D})\bar{T}_1$.
Let $\bar{P}$ denote the walk produced by starting $\Upsilon$ with $e$.
Return the input portion of $\bar{Q} = \bar{D}/\lambda(s_1, \bar{D})\bar{T}_1 label(\bar{P})$.
```

Proof : We know that the input portion of $\bar{Q}$ is a checking sequence from Theorem 4, Proposition 7, and Proposition 8. The optimality of $\bar{Q}$ follows from Proposition 9 and the fact that the walks added to form $Aug'(M)$ from $Aug(M)$ contain no reset transitions. $\square$

We can now consider the time complexity of Algorithm 4.

Proposition 10 *For an FSM with* n *states and* p *inputs, Algorithm 4 can be completed in time of* $O(pn^2 \log n)$.

Proof : The most computationally intensive parts of Algorithm 4 are the steps that produce $Aug(M)$ and that apply Algorithm 3. The first of these involves finding a min cost/max flow and for a digraph with v vertices and e edges this can be found in $O(ev \log v)$ (see, for example, [1]). Thus, this step takes time of $O(pn^2 \log n)$. Algorithm 3 has $O(n)$ iterations. Each iteration of Algorithm 3 involves finding two paths in a digraph with n vertices and pn edges; if a breadth-first search is used then each iteration takes time of $O(pn)$. Thus Algorithm 3 takes time of $O(pn^2)$ and the result follows. $\square$

Observe that it is possible for the walk produced by Algorithm 4 to end with a reset followed by connecting edges from E_{con}. If this is the case then the final reset can be eliminated from the checking sequence.

We now make some final observations regarding the proposed method. This assumed that the resets are implemented correctly and so are not included in the input alphabet X. If the resets are not known to be reliable then it is necessary to test these by following each by $\bar{D}/\lambda(s_1, \bar{D})\bar{T}_1$. This can be achieved by making the following changes to the algorithm: replace X by $X \cup \{r\}$, add the set $E_r = \{(v_i, v_1', r/-)|1 \leq i \leq n\}$ to the digraph $TestG(M)$ and include E_r in the set of required edges. The proposed algorithm uses distinguishing sequences, as is usual in checking sequence generation. Instead, it is possible to use adaptive distinguishing sequences and these provide a number of benefits. However, this is a topic of future work.

5 Experimental results

The proposed algorithm is parameterised by the α'-sequences and thus by the $\bar{T}_i$, $1 \leq i \leq n$. As explained in Section 4, we propose the heuristic of using empty transfer sequences. This is because the use of empty transfer sequences provides the optimisation algorithm with greater flexibility in choosing a walk that follows the test of a transition. In this section we report on the results of experiments, with randomly generated FSMs, that investigated the following questions:

1. How good are the results if we produce the α'-sequences using empty transfer sequences? This question concerns the effectiveness of the proposed heuristic of using empty $\bar{T}_i$.
2. What impact does the choice of transfer sequences have on the number of resets in the resultant checking sequence (what is the variability)? Here we are interested in the effect of the choice of transfer sequences since we want to know how robust our method is to a suboptimal choice of the $\bar{T}_i$.
3. How do the results compare with those produced using a method that does not attempt to minimise the number of resets and instead aims to minimise the checking sequence length? We consider this since we want to know whether the process of attempting to minimise the number of resets does actually reduce the number of resets.

The FSMs were randomly generated by inputting the number of states (n), the number of inputs (p) and the number of outputs (q) and for each state s and input x, randomly choosing the end state s' and output y. For each FSM M produced in this way we only kept M if it had a distinguishing sequence, was minimal and initially connected, and was not strongly connected. In order to allow a fair comparison between the proposed method and that described in [18] one small change was made to each of the two methods.

1. The method in [18] finds a walk that goes through the required set of edges, rather than a tour. The use of a walk can lead to shorter checking sequences, since there is no need to return to the initial state. We adapted the proposed method so that it produces a walk rather than a tour in order to avoid biasing the measurements of checking sequence length against it[6].
2. The method in [18] makes not attempt to avoid the use of resets in the $\bar{T}_i$. Instead, in the experiments when random $\bar{T}_i$ are generated for [18] we avoid the inclusion of resets in order to avoid biasing the experiments against the method of [18].

While this paper is concerned with minimising the number of resets used, we wanted to investigate the impact of this minimisation on the length of the resulting checking sequences. Thus for each checking sequence produced we recorded its length as well as the number of resets it contained. This also allowed us to compare the length of the checking sequence produced by the proposed method with one that aims to minimise the checking sequence length, not the number of resets [18]. For each FSM used in the experiments we did the following:

1. We produced a checking sequence using α'-sequences with empty transfer sequences and recorded the checking sequence length and the number of reset transitions included in the checking sequence.

[6] It is straightforward to change the proposed algorithm in order to achieve this. However, the use of a tour is described in this paper since it simplifies the exposition and this has no impact on the number of resets used, since we do not count the last reset (if any).

Table 1 The number of resets in checking sequences with empty $\bar{T}_i$

FSM	Number of states	Alphabet size	Number of resets MR	Number of resets M06	Mult factor
1	25	3	3	16	5.33
2	25	3	3	9	3
3	25	3	3	18	6
4	25	3	6	12	2
5	25	3	9	12	1.33
6	50	5	5	24	4.8
7	50	5	5	10	2
8	50	5	5	15	3
9	50	5	5	17	3.4
10	50	5	5	23	4.6
11	75	7	7	10	1.43
12	75	7	7	24	3.43
13	75	7	7	15	2.43
14	75	7	7	28	4
15	75	7	7	21	3
16	100	10	10	39	3.9
17	100	10	10	19	1.9
18	100	10	10	32	3.2
19	100	10	10	20	2
20	100	10	10	16	1.6

2. We randomly generated transfer sequences that did not contain reset transitions and produced α'-sequences using this. Given state s_i the transfer sequence $\bar{T}_i$ was randomly chosen in the following way: randomly select state s_j that can be reached from s_i without using reset transitions and let $\bar{T}_i$ be a minimum length path from s_i to s_j that contains no reset transitions. We produced a checking sequence, using these α'-sequences, and determined its length and the number of reset transitions it contained. For each FSM this process was repeated 100 times with independently randomly selected transfer sequences.

A total of 20 FSMs were randomly generated. In Table 1, the first three columns show the FSM number, the number of states of the FSM, and the size of the input and output alphabets[7] respectively. This is followed by two with MR denoting the proposed method, modified to use walks rather than tours, and M06 denoting the modified [18]. The two columns give the number of resets produced using empty $\bar{T}_i$. A final column gives the number of resets produced with $M06$ divided by the number of resets produced with MR. The number of resets are also shown in the graph in Figure 6.

We observe from Table 1 and Figure 6 that MR's number of resets for empty $\bar{T}_i$ is always less than that produced using M06 and empty $\bar{T}_i$. Table 2 reports the results of experiments with the same FSMs but using randomly generated $\bar{T}_i$. The first column gives the FSM number and this is followed by columns giving number of resets. Again, MR denotes the proposed method, modified to use walks rather than tours, and M06 denotes the modified [18]. There are four pairs of columns that report the number of resets in the checking sequences produced in 100 experiments with randomly generated $\bar{T}_i$: columns 2 and 3 give the minimum number of resets, columns 4 and 5 give the mean, while columns 7 and 8 give the maximum. Column 6 gives the ratio between the

[7] In all cases the input and output alphabets had the same size.

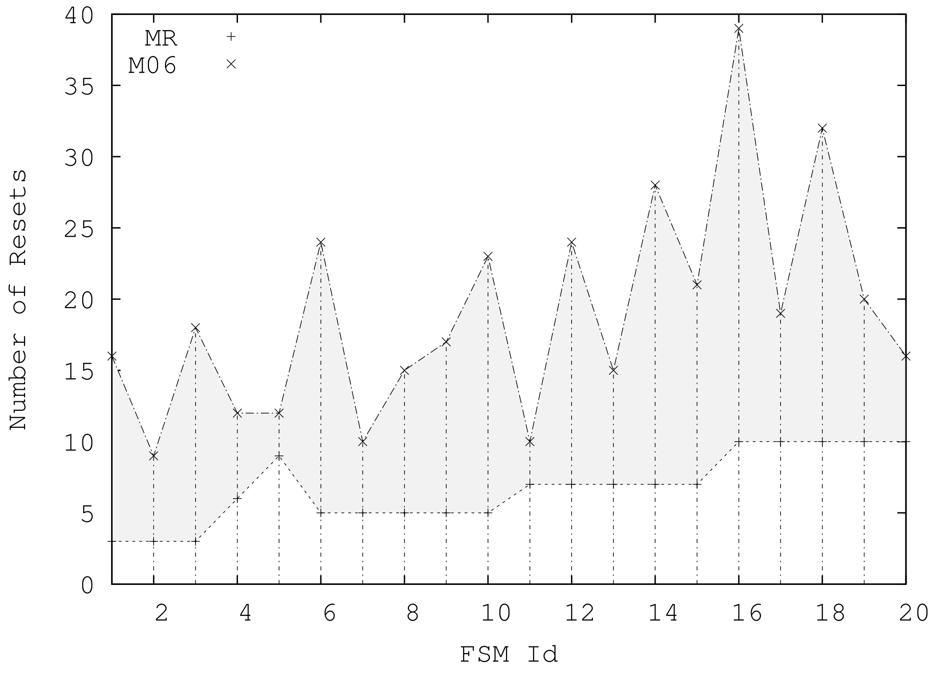

Fig. 6 The number of resets with empty $\bar{T}_i$

mean number of resets using M06 and the mean number of resets using MR. The last two columns give the number of checking sequences that ended in a reset in the 100 experiments for each method: such resets can be removed. The mean number of resets are shown in Figure 7.

The first observation to make relates to the heuristic of using empty $\bar{T}_i$ for the proposed method. Here, in every case the proposed method did not find a checking sequence with fewer resets than that produced using empty $\bar{T}_i$. From Table 1 we see that MR's number of resets for empty $\bar{T}_i$ is always less then that produced using M06 and empty $\bar{T}_i$. In addition, as shown in Figure 7, MR's mean number of resets for random $\bar{T}_i$ is consistently lower than that of M06. It is interesting to note that when we applied the method of [18] 100 times and used the checking sequence with fewest resets we obtained a checking sequence with the same number of resets as that produced using the proposed method with empty $\bar{T}_i$. In addition, the proposed method always included the same number of resets when random $\bar{T}_i$ were used while there is much more variability in the method of [18] when considering the number of resets. For example, with FSM 16 the minimum number of resets in a checking sequence found by the method of [18] was 10 but with empty $\bar{T}_i$ it produced a checking sequence with 39 resets and the experiments produced a checking sequence with 66 resets. These experiments suggest that, as would be expected, the proposed method is better at producing checking sequences with few resets than the method of [18].

Table 2 The number of resets with randomly generated $\bar{T}_i$

FSM	Min MR	Min M06	Mean MR	Mean M06	Mult factor	Max MR	Max M06	♯ ending in reset MR	♯ ending in reset M06
1	3	3	3.00	8.38	2.79	3	22	0	0
2	3	3	3.00	7.83	2.61	3	16	0	1
3	3	3	3.00	8.01	2.67	3	21	0	0
4	6	6	6.00	12.72	2.12	6	31	0	2
5	9	9	9.00	15.02	1.67	9	37	0	3
6	5	5	5.00	14.04	2.81	5	33	0	1
7	5	5	5.00	12.33	2.47	5	37	0	1
8	5	5	5.00	12.59	2.52	5	27	0	2
9	5	5	5.00	11.76	2.35	5	24	0	1
10	5	5	5.00	12.69	2.54	5	27	0	1
11	7	7	7.00	13.84	1.98	7	35	0	7
12	7	7	7.00	16.29	2.33	7	51	0	7
13	7	7	7.00	15.36	2.19	7	33	0	2
14	7	7	7.00	14.67	2.10	7	49	0	1
15	7	7	7.00	17.91	2.56	7	36	0	0
16	10	10	10.00	18.42	1.84	10	66	0	1
17	10	10	10.00	18.73	1.87	10	46	0	1
18	10	10	10.00	17.36	1.73	10	45	0	0
19	10	10	10.00	19.63	1.96	10	42	0	1
20	10	10	10.00	19.41	1.94	10	55	0	0

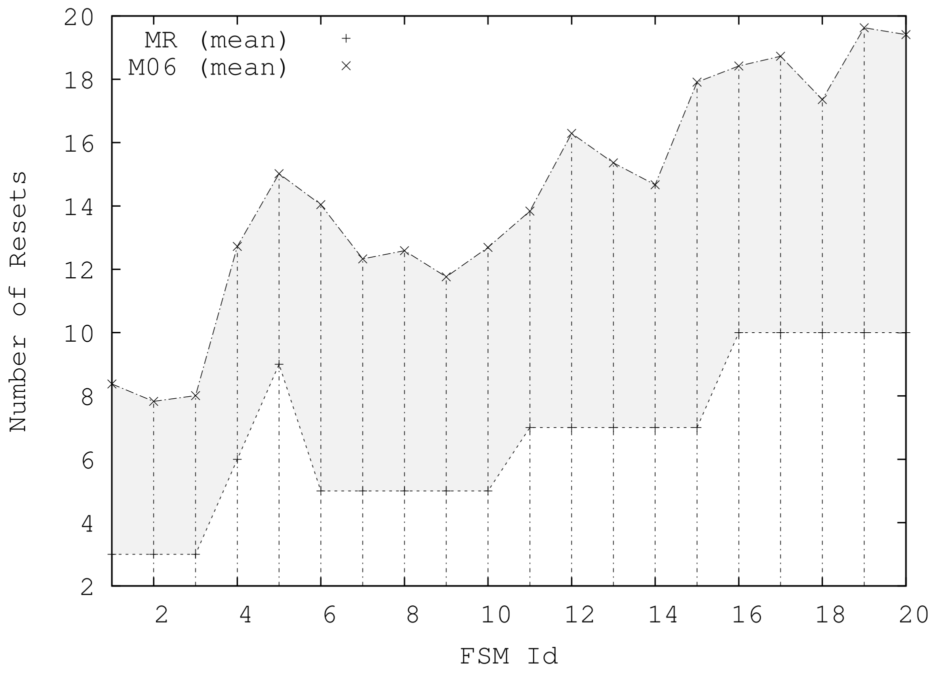

Fig. 7 The mean number of resets with random $\bar{T}_i$

Table 3 Checking sequence length with empty $\bar{T}_i$

FSM	Number states	Alphabet size	Empty T_i MR	Empty T_i M06	Difference
1	25	3	672	671	1
2	25	3	671	665	6
3	25	3	727	717	10
4	25	3	705	699	6
5	25	3	572	572	0
6	50	5	1717	1708	9
7	50	5	1668	1668	0
8	50	5	1720	1715	5
9	50	5	1746	1744	2
10	50	5	1734	1729	5
11	75	7	3461	3461	0
12	75	7	3461	3454	0
13	75	7	3393	3393	0
14	75	7	3432	3432	0
15	75	7	3383	3383	0
16	100	10	6279	6279	0
17	100	10	6198	6191	7
18	100	10	6185	6185	0
19	100	10	6114	6114	0
20	100	10	6284	6284	0

While the aim of the proposed algorithm is to minimise the number of resets used, we often also want a short checking sequence. Table 3 reports on the lengths of the checking sequences produced in the experiments with empty $\bar{T}_i$. From Table 3, we observe that MR's length of checking sequence for empty $\bar{T}_i$ is very similar to that of M06 and in half of the cases it is identical. The largest difference in checking sequence length is 10 and this is for checking sequences of length greater than 700. Despite these similarities in length, we have seen that there are considerable differences in the number of resets in these sequences.

Table 4 gives the results for checking sequences with randomly generated $\bar{T}_i$. Again, the results for MR and MR06 are similar when considering minimum length, mean length and maximum length. In fact, the largest difference is for the minimum length checking sequences for FSM 10 and this is just over 10%. If instead we consider the mean figures, the largest difference is less than 3%. The results in Tables 3 and 4 also show that as well as minimising the number of resets, the choice of empty $\bar{T}_i$ leads to the shortest checking sequences produced for each FSM. In the experiments the proposed method produced checking sequences of similar length to those of [18] suggesting that the process of minimising the number of resets has relatively little impact on the overall checking sequence length.

The proposed algorithm can be applied with strongly connected FSMs and so we ran experiments with six such FSMs. Tables 5 and 6 shows the number of resets in the checking sequences returned, MR always returning checking sequences with no resets. These are illustrated in Figures 8 and 9 respectively. In contrast, in all cases M06 included resets for some choice of $\bar{T}_i$ and in half of the cases it included resets when using empty $\bar{T}_i$. The lengths of the checking sequences are given in Tables 7 and 8, which again shows that MR produced checking sequences of a similar length to those returned by M06.

Table 4 Checking sequence length with randomly generated $\bar{T}_i$

FSM	min MR	min M06	mean MR	mean M06	max MR	max M06
1	879	854	936.31	915.86	994	980
2	862	840	912.90	896.94	970	955
3	874	841	931.69	915.32	987	976
4	861	825	931.14	904.60	991	972
5	727	697	786.94	768.58	843	817
6	2317	2271	2404.04	2370.71	2506	2464
7	2277	2253	2365.67	2344.02	2471	2503
8	2288	2234	2381.48	2359.99	2472	2501
9	2316	2222	2381.20	2352.43	2486	2450
10	2460	2230	2378.34	2352.41	2460	2447
11	4843	4480	4661.94	4627.62	4843	4747
12	4802	4479	4653.19	4612.32	4802	4750
13	4487	4467	4662.76	4643.34	4826	4816
14	4474	4492	4633.16	4607.13	4752	4782
15	4536	4495	4662.10	4629.26	4811	4864
16	8130	8096	8332.10	8284.05	8520	8559
17	8083	8021	8316.74	8266.96	8554	8463
18	8135	7988	8327.46	8277.78	8548	8472
19	8165	8125	8341.07	8304.46	8523	8489
20	8193	8123	8345.36	8308.63	8516	8508

Table 5 Number of resets for connected FSMs with empty $\bar{T}_i$

FSM	Number of states	Number of inputs	Empty T_i MR	Empty T_i M06
S1	3	2	0	0
S2	5	3	0	0
S3	10	5	0	10
S4	15	4	0	4
S5	20	5	0	0
S6	25	7	0	7

Table 6 Number of resets for connected FSMs with randomly generated $\bar{T}_i$

FSM	Min MR	Min M06	Mean MR	Mean M06	Max MR	Max M06
S1	0	0	0	0.61	0	2
S2	0	0	0	1.88	0	9
S3	0	0	0	2.58	0	15
S4	0	0	0	3.08	0	16
S5	0	0	0	3.04	0	18
S6	0	0	0	4.06	0	21

Table 7 Checking sequence length for connected FSMs with empty $\bar{T}_i$

FSM	Number of states	Number of inputs	Empty T_i MR	Empty T_i M06	Difference
S1	3	2	27	27	0
S2	5	3	75	75	0
S3	10	5	289	284	5
S4	15	4	363	355	8
S5	20	5	578	578	0
S6	25	7	1080	1080	0

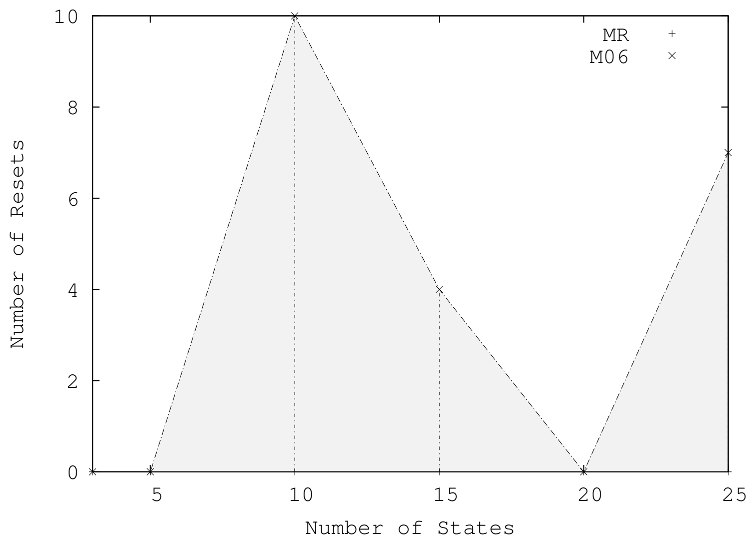

Fig. 8 The number of resets with empty $\bar{T}_i$

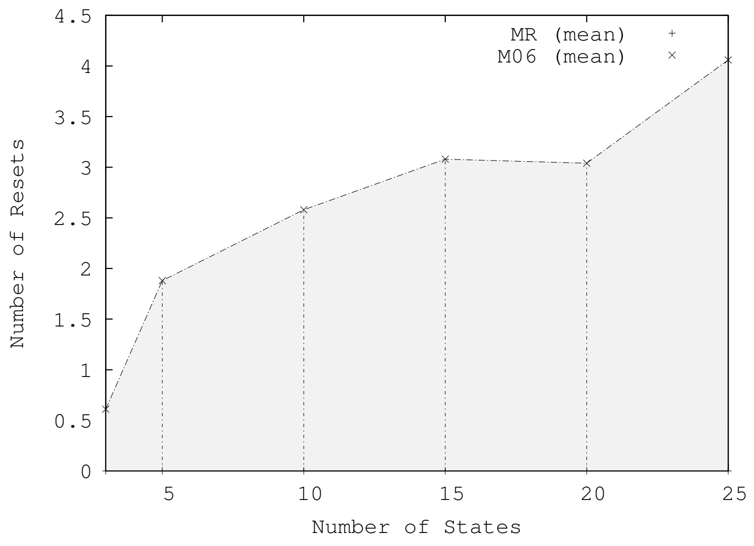

Fig. 9 The number of resets with random $\bar{T}_i$

Table 8 Checking sequence length for connected FSMs with randomly generated $\bar{T}_i$

FSM	Min MR	Min M06	Mean MR	Mean M06	Max MR	Max M06
S1	36	27	41.28	37.07	44	44
S2	76	76	91.16	90.60	122	119
S3	303	299	350.52	348.85	403	396
S4	444	424	500.60	491.68	604	573
S5	738	703	824.54	817.65	937	929
S6	1310	1263	1441.65	1429.81	1660	1668

6 Conclusions and Observations

A checking sequence for a finite state machine (FSM) M is an input sequence that is guaranteed to lead to a failure if the implementation under test (IUT) is faulty and has no more states than M. It is desirable to use a short checking sequence and there has thus been much interest in automatically generating such a checking sequence. However, in some situations the use of resets increases the cost of testing and reduces the expected effectiveness of the checking sequence and in such cases we may want to minimise the number of reset transitions used.

This paper investigated the problem of producing a checking sequence that has a minimum number of resets. It considered a class of checking sequences that is defined by recent checking sequence generation algorithms. The proposed algorithm returns a checking sequence that, amongst those in this class, has a minimum number of resets. For an FSM with n states and p inputs, the algorithm has time complexity of $O(pn^2 \log n)$. In contrast to other checking sequence generation algorithms, the approach given in this paper does not require the FSM to be strongly connected.

The proposed checking sequence generation algorithm is parameterised by a set of transfer sequences. This paper reported on experiments used to investigate the effectiveness of one heuristic: using empty transfer sequences. A total of 20 randomly generated FSMs were used in the experiments: for each a checking sequence was produced using empty $\bar{T}_i$ and checking sequences were produced using 100 randomly generated $\bar{T}_i$. In all of the experiments the checking sequence with empty $\bar{T}_i$ was both the shortest checking sequence and the checking sequence with fewest resets.

Experiments were used to compare the proposed method with a recent checking sequence generation method that aims to minimise the checking sequence length [18]. As expected, it was found that the proposed method was never outperformed by the algorithm of [18], when considering the number of resets in the checking sequence returned. In addition, the heuristic of using empty $\bar{T}_i$ appeared to be less effective with the method of [18]. The lengths of the checking sequences returned by the proposed method were similar to the lengths of the checking sequences returned by [18]. Similar results were obtained when the two methods were applied to completely specified FSMs. It should be remembered that the method of [18] requires us to solve an NP-hard optimisation problem while the proposed method requires low order polynomial time.

The checking sequence generated by the proposed algorithm is the input portion of $\bar{D}/\lambda(s_1, \bar{D})\bar{T}_1$ followed by the input portion of the label of a tour Υ started at the vertex reached from v_1 by a walk with label $\bar{D}/\lambda(s_1, \bar{D})\bar{T}_1$. If there is a walk $\bar{P}_1$ from state s_i to s_1 that contains no reset transitions, and Υ contains reset transitions, then we can eliminate one reset transition from the checking sequence. If it is possible to eliminate a reset transition when using non-empty transfer sequences, then it is also possible to eliminate a reset transition when using empty transfer sequences. Thus the observation, that it is sometimes possible to eliminate one reset transition, does not invalidate the experiments reported in Section 5, that investigated the effectiveness of using empty transfer sequences.

Acknowledgements The authors wish to thank Yaping Luo, Tuong Nguyen and Ji Chao Zhang for implementing the algorithms and performing the experiments described in this paper. We would also like to thank the anonymous referees for their extremely helpful comments and suggestions.

References

1. 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).
2. M. Barnett, W. Grieskamp, L. Nachmanson, W. Schulte, N. Tillmann, and M. Veanes. Towards a tool environment for model-based testing with AsmL. In *Formal Approaches to Testing*, volume 2931 of *Lecture Notes in Computer Science*, pages 252–266, Montreal, Canada, 2003. Springer-Verlag.
3. R. V. Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools.* Addison-Wesley, 1999.
4. B. Broekman and E. Notenboom. *Testing Embedded Software.* Addison-Wesley, London, 2003.
5. T. S. Chow. Testing software design modelled by finite state machines. *IEEE Transactions on Software Engineering*, 4:178–187, 1978.
6. Adenilso da Silva Simão and Alexandre Petrenko. Generating checking sequences for partial reduced finite state machines. In *20th IFIP TC 6/WG 6.1 International Conference Testing of Software and Communicating Systems, 8th International Workshop on Formal Approaches to Testing of Software (TestCom/FATES 2008)*, volume 5047 of *Lecture Notes in Computer Science*, pages 153–168. Springer, 2008.
7. J. Dick and A. Faivre. Automating the generation and sequencing of test cases from model-based specifications. In *FME '93, First International Symposium on Formal Methods in Europe*, pages 268–284, Odense, Denmark, 19-23 April 1993. Springer-Verlag, Lecture Notes in Computer Science 670.
8. A. Y. Duale and M. U. Uyar. A method enabling feasible conformance test sequence generation for EFSM models. *IEEE Transactions on Computers*, 53(5):614–627, 2004.
9. E. Farchi, A. Hartman, and S. Pinter. Using a model-based test generator to test for standard conformance. *IBM systems journal*, 41(1):89–110, 2002.
10. Mario Friske and Bernd-Holger Schlingloff. Improving test coverage for UML state machines using transition instrumentation. In *26th International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, volume 4680 of *Lecture Notes in Computer Science*, pages 301–314. Springer, 2007.
11. S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite state models. *IEEE Transactions on Software Engineering*, 17(6):591–603, 1991.
12. G. Gonenc. A method for the design of fault detection experiments. *IEEE Transactions on Computers*, 19:551–558, 1970.
13. D. Harel and M. Politi. *Modeling reactive systems with statecharts: the STATEMATE approach.* McGraw-Hill, New York, 1998.
14. M. Haydar, A. Petrenko, and H. Sahraoui. Formal verification of web applications modeled by communicating automata. In *Formal Techniques for Networked and Distributed Systems (FORTE 2004)*, volume 3235 of *Springer Lecture Notes in Computer Science*, pages 115–132, Madrid, September 2004. Springer-Verlag.
15. 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.

16. R. M. Hierons. Minimizing the number of resets when testing from a finite state machine. *Information Processing Letters*, 90(6):287–292, 2004.
17. R. M. Hierons and H. Ural. Reduced length checking sequences. *IEEE Transactions on Computers*, 51(9):1111–1117, 2002.
18. Robert M. Hierons and Hasan Ural. Optimizing the length of checking sequences. *IEEE Transactions on Computers*, 55(5):618–629, 2006.
19. 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.
20. ITU-T. *Recommendation Z.100 Specification and description language (SDL)*. International Telecommunications Union, Geneva, Switzerland, 1999.
21. D. Lee and M. Yannakakis. Principles and methods of testing finite-state machines - a survey. *Proceedings of the IEEE*, 84(8):1089–1123, 1996.
22. J. L. Lenstra and Rinnoy Khan. On general routing problems. *Networks*, 6:273–280, 1976.
23. 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.
24. E. P. Moore. Gedanken-experiments. In C. Shannon and J. McCarthy, editors, *Automata Studies*. Princeton University Press, 1956.
25. H. Ural, X. Wu, and F. Zhang. On minimizing the lengths of checking sequences. *IEEE Transactions on Computers*, 46(1):93–99, 1997.
26. M. P. Vasilevskii. *Failure Diagnosis of Automata. Cybernetics*. Plenum Publishing Corporation, 1973.
27. M. Yao, A. Petrenko, and G. v. Bochmann. Conformance testing of protocol machines without reset. In *Protocol Specification, Testing and Verification, XIII (C-16)*, pages 241–256. Elsevier (North-Holland), 1993.