Article:

Turker, U.C., Hierons, R. orcid.org/0000-0002-4771-1446 and Jourdan, G.-V. (2021) Minimizing characterizing sets. Science of Computer Programming, 208. 102645. ISSN 0167-6423

https://doi.org/10.1016/j.scico.2021.102645

Manuscript

Minimizing Characterizing sets

Uraz Cengiz Türker[a,*], Robert M. Hierons[b], Guy-Vincent Jourdan[c]

[a] *University of Leicester, School of Informatics, Leicester, UK*
[b] *The University of Sheffield, Department of Computer Science, Sheffield, UK*
[c] *School of Electrical Engineering and Computer Science, Faculty of Engineering, University of Ottawa, Ottawa, CA*

Abstract

A characterizing set (CS) for a deterministic finite state machine (FSM) M is a set of input sequences that, between them, separate (distinguish) all of the states of M. CSs are used within several test generation techniques that return test suites with guaranteed fault detection power. The number of input sequences in a CS directly affects the cost of applying the resultant test suite. In this paper, we study the complexity of decision problems associated with deriving a smallest CS from an FSM, showing that checking the existence of a CS with K sequences is PSPACE-complete. We also consider the length of a CS, which is the sum of the lengths of the input sequences in the CS. It transpires that the problem of deciding whether there is a CS with length at most K is NP-complete. Motivated by these results, we introduce a heuristic to construct a CS, from a deterministic FSM, with the aim of minimizing the number of input sequences. We evaluated the proposed algorithm by assessing its effect when used within a classical test generation algorithm (the W-method). In the evaluation, we used both randomly generated FSMs and benchmark FSMs. The results are promising, with the proposed algorithm reducing the number of test sequences by 37.3% and decreasing the total length of the test suites by 34.6% on average.

*Corresponding author

Email addresses: u.c.turker@leicester.ac.uk (Uraz Cengiz Türker), r.hierons@sheffield.ac.uk (Robert M. Hierons), gjourdan@uottawa.ca (Guy-Vincent Jourdan)

Keywords: Model-based testing, Characterizing set, Complexity

1. Introduction

Testing is an indispensable aspect of a development-cycle for any kind of system. However, manual testing is typically expensive and error prone. This has led to significant interest in test automation and, in particular, Model Based Testing (MBT). MBT techniques and tools use behavioural models and usually operate on either a finite state machine (FSM), extended finite state machine (EFSM) or labelled transition system (LTS) that defines the semantics of the model. This paper concentrates on testing from a (deterministic) FSM. There has been significant interest in automating testing based on an FSM model in areas such as sequential circuits [1], lexical analysis [2], software design [3], communication protocols [3, 4, 5, 6, 7, 8, 9, 10], object-oriented systems [11], and web services [12, 13, 14, 15]. Such techniques have also been shown to be effective when used in large industrial projects [16].

The literature contains many formal methods to automatically generate test suites (sets of test sequences) from FSM models (*specifications*) of systems [9, 17, 18, 19, 20, 21, 22, 23, 24]. These methods represent testing as running *fault detection experiments* [25]. Formal methods for generating fault detection experiments are based on particular types of sequences that are derived from the specification M. Different test techniques use different types of sequence but we focus on the use of a *characterizing set (W-set)*, to identify the current state of the implementation, motivated by the fact that every (minimal, deterministic) FSM has a W-set. A W-set is a set of input sequences such that for any pair (s,s') of distinct states of M there exists an input sequence $\bar{x}$ in the W-set that separates (distinguishes) s and s' [26].

1.1. Motivation and Problem Statement

Characterizing sets are widely used in test generation. For example, they are used in the classical *W-method* [3, 27], which has two steps: state recognition and transition verification. Importantly, test generation involves separately following a particular set of input sequences by *every* input sequence from the characterizing set $\mathcal{W}$ used. In testing, the system under test is reset before an input sequence is applied, ensuring that the input sequences are applied in the same state of the system under test.

Characterizing sets are used in some other test generation techniques. For example, the well known HSI-method [28][1] requires *harmonized state identifiers* (HSIs) to construct test sequences from a given possibly non-deterministic FSM M. Construction of HSIs requires one to harmonize the elements of a W-set of M. That is, in order to construct HSIs for M, one first has to construct a W-set [28].

A further example, of the use of characterizing sets, can be found in automated model learning [29]. Here, the W-method is used to decide whether a given hypothesis (FSM) is correct or not [30]. Since testing whether the hypothesis holds (the oracle problem) is computationally expensive it has been referred to as the bottleneck for learning models from complex systems [31]. Therefore, any method that generates compact W-sets could help such learning methods to scale to larger problems.

It is clear that the size of the test suite returned by a test generation algorithm that use a characterizing set $\mathcal{W}$ is directly affected by the number of input sequences that are in $\mathcal{W}$. As explained above, if the W-set $\mathcal{W}$ contains k input sequences then test generation techniques such as the W-method will (separately) follow a set of input sequences by all k input sequences in $\mathcal{W}$. In addition, there are cases where the reset between test sequences is particularly time consuming or expensive since, for example, it may require a system to be reconfigured or may require manual intervention. As a result, there has been interest in constructing test generation techniques that return test suites with the minimum number of resets (and so test se-

[1] Note that in [28] there are two algorithms: the HSI-method for constructing the test suite, and the HSI-algorithm for constructing state identifiers. Therefore we use *HSI-method* and *HSI-algorithm* to denote the method and the algorithm respectively.

quences) [3, 9, 21, 22, 32, 27, 33, 34, 35, 36, 28, 37, 38, 39].

This paper revisits the long-standing problem of finding a smallest characterizing set for an FSM [26]. The aim is to reduce the number of test sequences, within a test suite, by minimizing the number of elements in W-sets. At a high-level, this paper makes two main contributions. First, we determine the computational complexity of decision problems associated with the generation of a smallest characterizing set, a problem that has been open for more than half a century (see, [26]). We prove that the problem of deciding whether an FSM has a W-set that contains at most K test sequences is PSPACE-complete. In addition, we prove that the problem of deciding whether an FSM has a W-set $\mathcal{W}$, such that the sum of the lengths of the test sequences in $\mathcal{W}$ is at most K, is NP-complete. Complexity is in terms of the size of the problem description but it will transpire that the most important parameter is the number (n) of states of the FSM M, since it is possible to place upper bounds on 'interesting' values of K, with these upper bounds being polynomials in n. For example, every (minimal) FSM with n states has a characterizing set that contains at most $n-1$ input sequences. We then give a heuristic that aims to generate a small characterizing set and evaluate this heuristic through experiments with both randomly generated FSMs and benchmark FSMs. To the best of our knowledge, this is the first attempt to devise heuristics for generating a small characterizing set. In this paper, we therefore focus on the following problems, along with their complexities (these problems are formally defined in Section 2).

1. The MinSize W-set problem: given a deterministic FSM M and positive integer K, does M have a W-set that contains no more than K input sequences.
2. The MinLength W-set problem: given a deterministic FSM M and positive integer K, does M have a W-set $\mathcal{W}$ such that the sum of the lengths of the input sequences in $\mathcal{W}$ does not exceed K.

In naming the problems, we differentiate between the *size* of a W-set $\mathcal{W}$ (the number of input sequences in set $\mathcal{W}$) and the *length* of a W-set $\mathcal{W}$ (the sum of the lengths of the input sequences in $\mathcal{W}$). As explained above, the size and length of a W-set are both of interest.

1.2. Results

We show that the MinSize W-set problem is PSPACE-complete. We also prove that the MinLength W-set problem is NP-complete.

We also introduce a heuristic that uses a breadth first search strategy, on input sequences, to find a relatively small W-set for deterministic FSMs. We evaluated the heuristic through experiments with randomly generated and benchmark FSMs. In these experiments we assessed the impact of using the heuristic for W-set generation, when the W-sets are used in the W-method. We found that the heuristic can reduce the number of test sequences (for the W-method) by 37.3% and can decrease the total length of the test suite by 36.4% on average.

This paper extends a previous conference paper [40] in the following ways. First, previous work only considered the complexity of the MinSize W-set problem; this paper extends this by also considering the overall W-set length (the MinLength W-set problem). Second, we extended the experimental evaluation in several ways. First, we report additional results, such as the overall W-set size, test suite size, and test suite length; previously, only ratios were reported. Second, we report the results of an analysis that demonstrates that the differences observed are statistically significant (except for the smallest FSMs) and that there is a large effect-size. Finally, we extended the set of benchmark FSMs used from five to sixteen.

1.3. Practical Implications of our Results and Future Directions

W-sets are used in many formal methods that automate the generation of test suites, therefore methods that use these sequences will directly benefit from the research in this paper. In order to assess the effect of deriving smaller W-sets, future work will extend the empirical evaluation to other test suite generation algorithms and also learning.

1.4. Structure of the paper

This paper is structured as follows. In the next section, we provide the terminology and notation used throughout the paper. In Section 3, we prove that the MinSize W-set problem is PSPACE-complete and that the MinLength W-set problem is NP-complete. This section is then followed by a section (Section 4) that introduces the proposed algorithm. In Section 5, we describe the experiments performed to evaluate the proposed algorithm and the results of these experiments. Section 6 then describes related work and we conclude the paper by providing some future directions in Section 7.

2. Preliminaries

2.1. Finite State Machines (FSMs)

An FSM M is defined by a tuple (S, s_0, X, Y, h) where $S = \{s_1, s_2, \ldots, s_n\}$ is a finite set of states, $s_0 \in S$ is the initial state, $X = \{x_1, x_2, \ldots, x_p\}$ and $Y = \{y_1, y_2, \ldots, y_q\}$ are finite sets of inputs and outputs, and $h : S \times X \times Y \times S$ is the set of transitions (the transition relation). Tuple $\tau = (s, x, y, s') \in h$ is a *transition* that has *starting state* s, *ending state* s', and *label* x/y. We can interpret τ as meaning that if M receives input x when in state s then it can output y and move to state s'. M can then receive another input when in state s'.

Given a set X we let X^* denote the set of finite sequences of elements of X and let X^k denote the set of sequences in X^* of length k. We use ε to denote the empty sequence and given sequences $\bar{x}$ and $\bar{x}'$, $\bar{x}\bar{x}'$ denotes the concatenation of $\bar{x}$ and $\bar{x}'$. It is possible to extend h, to a relation $h^* : S \times X^* \times Y^* \times S$ that is the smallest relation such that: 1) for all $s \in S$, $(s, \varepsilon, \varepsilon, s) \in h^*$; and 2) if $(s_1, \bar{x}, \bar{y}, s_2) \in h^*$ and $(s_2, x, y, s_3) \in h$ then $(s_1, \bar{x}x, \bar{y}y, s_3) \in h^*$. If $(s, \bar{x}, \bar{y}, s') \in h^*$ then this means that input sequence $\bar{x}$ can take the FSM from state s to state s' while producing output sequence $\bar{y}$.

An FSM M can be represented by a directed graph in which nodes represent states of M and edges represent transitions of M. Figure 1 gives an example of a directed graph that represents an FSM that will be called M_1. Here, for example, the edge from the vertex with label s_4 to the vertex with label s_1 represents the transition (s_4, x_2, y_1, s_1). In the diagram, if a list of input/output pairs is given on an edge then this correspond to multiple transitions that have the same starting and ending states. For example, the edge from the node with label s_1 to the node with label s_2 represents transitions (s_1, x_1, y_1, s_2) and (s_1, x_3, y_1, s_2).

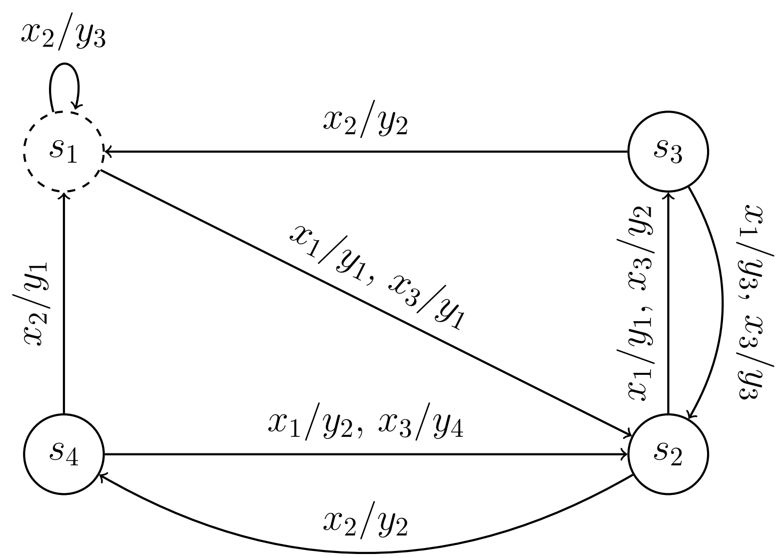


Figure 1: FSM M_1. Note that the initial state is highlighted with a dashed line.

FSM M is said to be *completely-specified* if for every state s and input x there is at least one transition $\tau = (s, x, y, s') \in h$ with starting state s and input x. It is straightforward to check that M_1 is completely-specified. If M is not completely-specified then it is *partial*. In this paper we only consider completely-specified FSMs, motivated by two factors. First, the vast majority of FSM-based test generation algorithms assume that the FSM specification is completely-specified. Second, if the FSM specification M is not completely-specified then it is often possible to complete M by adding, for example, self-loops or an error state. FSM M is said to be *deterministic* if for every state s and input x, M has only one transition $\tau = (s, x, y, s') \in h$ with input x and starting state s. For example, M_1 is deterministic. In this paper we only consider deterministic FSMs. Although not all FSMs are deterministic, deterministic FSMs have been the main focus of FSM-based test generation.

A *walk* ρ of M is a sequence $(s_1, x_1, y_1, s_2)(s_2, x_2, y_2, s_3) \ldots (s_k, x_k, y_k, s_{k+1})$ of consecutive transitions and ρ has starting state $start(\rho) = s_1$, ending state $end(\rho) = s_{k+1}$, and $label(\rho) = x_1/y_1 \ldots x_k/y_k$. The behaviour of an FSM M is defined in terms of the labels of walks leaving the initial state; such labels of walks are called *traces*. For example, $\rho_1 = (s_1, x_1, y_1, s_2)(s_2, x_2, y_2, s_4)$ (s_4, x_2, y_1, s_1) is a walk of M_1 (Figure 1); ρ_1 has *starting state* s_1, *ending state* s_1, and *label* $x_1/y_1\ x_2/y_2\ x_2/y_1$. Here $\sigma = x_1/y_1 \ldots x_k/y_k$ is an input/output sequence, sometimes called a *trace*, that has *input portion* $in(\sigma) = x_1 \ldots x_k$ and *output portion* $out(\sigma) = y_1 \ldots y_k$. An FSM M with initial state s_0 is *initially connected* if for every state s of M there exists some walk that has starting state s_0 and ending state s. It is straightforward to see that M_1 is initially connected.

2.2. FSM behaviour

FSM M defines the language $L(M)$ of labels of walks with starting state s_0 and $L_M(s)$ denotes the language obtained if we make s the initial state. Thus, $L_M(s) = \{x_1/y_1 \ldots x_m/y_m \in X^*/Y^* | \exists s_1, \ldots, s_{m+1}. s_1 = s \wedge \forall 1 \leq i \leq m. (s_i, x_i, y_i, s_{i+1}) \in h\}$. For example, $x_1/y_1\ x_1/y_2 \in L_{M_1}(s_1)$. Given state s and input sequence $\bar{x}$ we use $M(s, \bar{x}) = \{\sigma \in L_M(s) | i(\sigma) = \bar{x}\}$ to denote the set of traces in $L_M(s)$ that have input portion $\bar{x}$. For example, in M_1 we have that $M_1(s_1, x_1x_2) = \{x_1/y_1\ x_2/y_2\}$ and $M_1(s_3, x_1x_2) = \{x_1/y_3\ x_2/y_2\}$. Given $S' \subseteq S$, $L_M(S') = \cup_{s \in S'} L_M(s)$ is the set of traces that can be produced if the initial state of M is in S'. In addition, $M(S', \bar{x}) = \cup_{s \in S'} M(s, \bar{x})$ denotes the set of traces that can result from applying $\bar{x}$ to a state in S'.

States s, s' of M are *equivalent* if $L_M(s) = L_M(s')$ and FSMs M and N are *equivalent* if $L(M) = L(N)$. FSM M is *minimal* if no two of its states are equivalent. As usual, in this paper we only consider minimal FSMs. This is not a restriction since an FSM can be rewritten to an equivalent minimal FSM in polynomial time using any technique that minimizes a deterministic finite automaton. To summarise, we consider deterministic FSMs that are minimal and completely-specified.

We also make the usual assumption, when using techniques such as the W-method, that the system under test has a reliable reset: some process that is guaranteed to return the system under test to its initial state. A reliable reset is required in order to ensure that the test sequences are all applied in the same state of the system under test.

2.3. Characterizing sets

The notion of a characterizing set is typically defined in terms of what it means for an input sequence $\bar{x}$ to separate two states of an FSM M.

Definition 1. *Input sequence $\bar{x}$* separates *states s and s' of FSM M if and only if $M(s, \bar{x}) \cap M(s', \bar{x}) = \emptyset$.*

We now define *W-Sets*.

Definition 2. *Given FSM M, $W \subseteq X^*$ is a* W-Set *for M if for any pair of distinct states of M there exists a sequence $\bar{x} \in W$ such that $\bar{x}$ separates the pair, i.e., $\forall s, s' \in S$, such that $s \neq s'$, there is some $\bar{x} \in W$ with $M(s, \bar{x}) \cap M(s', \bar{x}) = \emptyset$.*

Consider again the FSM M_1 in Figure 1. If we consider the set $\{x_1, x_2\}$ of inputs, the corresponding output responses for the states of M_1 are shown in Table 1. From this, one can see that every pair of distinct states of M_1 is separated by either x_1 or x_2 (or both). As a result, $\mathcal{W} = \{x_1, x_2\}$ is a W-set of M_1.

In the next section, we formalise and then explore the complexity of the following problem.

Definition 3. *Let M be a minimal deterministic completely specified FSM, and let K be a positive integer. In the MinSize W-set problem we are asked to decide whether there exists a W-set $\mathcal{W}$ for M such that $|\mathcal{W}| \leq K$.*

State	Response to x_1	Response to x_2
s_1	y_1	y_3
s_2	y_1	y_2
s_3	y_3	y_2
s_4	y_2	y_1

Table 1: Response to x_1 and x_2

We might also be interested in a W-set with smallest overall length, leading to the following problem in which $|w_i|$ is the length of the sequence w_i.

Definition 4. *Let M be a minimal deterministic completely specified FSM, and let K be a positive integer. In the MinLength W-set problem we are asked to decide whether there exists a W-set $\mathcal{W}$ for M such that $\sum_{w_i \in \mathcal{W}} |w_i| \leq K$.*

3. Complexity results

This section concerns the computational complexity of W-set decision problems.

3.1. Complexity of the MinSize W-set problem

In this section we start by showing that the problem is in PSPACE; we then prove that it is PSPACE-hard.

Proposition 1. *It is possible for a non-deterministic Turing Machine to decide in polynomial space whether a completely-specified deterministic FSM M has a W-set of size at most K.*

Proof. We will show how a non-deterministic Turing Machine can solve the problem in polynomial space. This Turing Machine will guess K input sequences, extending all K input sequences by one input in each iteration. It will keep tuples of the form (s, s', c) where $s, s' \in S$ and $c \in \{0, 1\}$. For each pair (s, s') of distinct states of M, the Turing Machine will start with K copies of

$(s, s', 0)$; one copy for each input sequence that the Turing Machine is to guess. As the process of guessing an input sequence progresses, the value of c will be changed to 1 if s and s' have been distinguished by the corresponding input sequence.

As mentioned, the Turing Machine will guess K input sequences $\bar{x}_0, \bar{x}_1, \ldots, \bar{x}_{K-1}$ in an iterative manner, extending all sequences by one input in each iteration. There are $n(n-1)/2$ pairs of states. Since M has n states and p inputs, encoding a state identifier requires $O(\log(n))$ space and encoding an input requires $O(\log(p))$ space. As a result, the Turing Machine requires polynomial space to store the tuples and the inputs guessed in the current iteration. At each iteration, K next inputs are guessed non-deterministically. Each guessed input x is used to update the corresponding tuples in the natural way (i.e. if $(s, x, y, s_1), (s', x, y', s'_1) \in h$ then $(s, s', c_b) \rightarrow (s_1, s'_1, c)$ where $c = 1$ if $c_b = 1$ or $y \neq y'$, and $c = 0$ otherwise).

Consider now how the Turing Machine terminates. First, it terminates with success if each pair (s, s') of distinct states has been distinguished; s and s' have been distinguished if for some guessed input sequence the tuple $(s, s', 0)$ has been transformed into some $(s_1, s'_1, 1)$. The Turing Machine also has a counter ctr, which is increased in each iteration, in order to ensure termination and we now give a bound that can be placed on this counter. If we consider the process of extending one of the input sequences $\bar{x}_i$ being guessed, in each iteration we obtain a sequence of tuples of the form (s, s', c); one for each $(s, s', 0)$ that we started with. Clearly, when looking for a bound on sequence length, it is sufficient to consider sequences $\bar{x}_i$ such that for every pair of prefixes $\bar{x}'_i$ and $\bar{x}''_i$ of $\bar{x}_i$, with $|\bar{x}'_i| < |\bar{x}''_i|$, we have that $\bar{x}'_i$ and $\bar{x}''_i$ define different sequences of tuples (otherwise, we can replace $\bar{x}''_i$ by $\bar{x}'_i$ to obtain a shorter sequence). There are $2n^2$ possible values for each tuple and $n(n-1)/2$ tuples and so at most $2^{n(n-1)/2} n^{n(n-1)}$ possible values for the tuples. Thus, the Turing Machine can terminate with failure if c exceeds this bound. As a result, the space required to store the counter is of $O(\log_2(2^{n(n-1)/2} n^{n(n-1)}))$ and so only polynomial space is required. The result therefore follows.

We now show that the MinSize W-set problem is PSPACE-complete.

Theorem 1. *The MinSize W-set problem for completely specified deterministic FSMs is* PSPACE-complete.

Proof. First, by Proposition 1 we know that the problem is in PSPACE. We now show that the problem is PSPACE-hard. Consider the case where $K = 1$. Then there is a W-set of size at most K for M if and only if there is a preset distinguishing sequence (PDS)[2] for M. Thus, any algorithm that can decide whether an FSM has a W-set of size K can also be used to decide whether an FSM has a PDS. Since the problem of deciding whether a deterministic FSM has a PDS is PSPACE-hard [41], we have that the problem of deciding whether an FSM has a W-set of size at most K is also PSPACE-hard. The result thus follows.

3.2. Complexity of the MinLength W-set problem

The motivation for the work in this paper is that a small W-set is likely to lead to a small test suite and here we are interested in how many input sequences are contained in the W-set. However, as discussed earlier, we might instead be interested in the overall length of a W-set $\mathcal{W}$: the sum of the lengths of the input sequences in $\mathcal{W}$. We now consider the complexity of this problem for deterministic FSMs.

We will first show that the problem is NP-hard by relating it to the Hitting Set Problem.

Definition 5. *Let us suppose that A is a finite set, $\{A_1, \ldots, A_k\}$ is a set of subsets of A, and K is an integer. The* Hitting Set Problem *is to decide whether there exists some subset A' of A, of size at most K, such that every A_i contains at least one element of A' ($A_i \cap A' \neq \emptyset$).*

The hitting set problem is known to be NP-complete [42]. We use this result to prove that the MinLength W-set problem is NP-hard.

[2] An input sequence $\bar{x}$ is a PDS for M if $\bar{x}$ distinguishes all of the states of M.

Proposition 2. *The MinLength W-set problem is NP-hard.*

Proof. Let us assume that we are given an instance of the Hitting Set Problem defined by set $A = \{a_1, \ldots, a_p\}$, set $\mathcal{A} = \{A_1, \ldots, A_n\}$ of subsets of A, and integer K. We will show how one can construct an FSM $M_{A,\mathcal{A}}$ such that $M_{A,\mathcal{A}}$ has a W-set of length at most K if and only if there is a solution to the instance of the Hitting Set Problem defined by A, $\mathcal{A}$, and K.

The FSM $M_{A,\mathcal{A}}$ will have one state s_j for each element A_j of $\mathcal{A}$ ($1 \leq j \leq n$). There is an additional state s_0, which is also the initial state. We will have output set Y that, for all $1 \leq i \leq n$ and $1 \leq j \leq p$, contains a unique output y_{ij}, as well as another output y_0. Thus, Y contains $np + 1$ outputs. For each $a_i \in A$ we will introduce a unique input x_i and we will add transitions so that x_i distinguishes state s_j from other states (including s_0) if and only if $a_i \in A_j$. This will be achieved through the following transitions ($1 \leq i \leq n, 1 \leq j \leq p$).

1. If $a_i \in A_j$ then $M_{A,\mathcal{A}}$ has transition (s_j, x_i, y_{ij}, s_j).
2. If $a_i \notin A_j$ then $M_{A,\mathcal{A}}$ has transition (s_j, x_i, y_0, s_j).

For all $1 \leq i \leq p$, $M_{A,\mathcal{A}}$ also has the transition (s_0, x_i, y_0, s_0).

Note that $M_{A,\mathcal{A}}$ is not initially connected as all its transitions loop. It is straightforward to extend this FSM to form an initially connected FSM by introducing additional input symbols and transitions that connect the initial state with all other states but that cannot be used for distinguishing the states of $M_{A,\mathcal{A}}$. Specifically, for all $1 \leq j \leq n$ we introduce a new input x'_j that reaches state s_j from the initial state. The transitions with input x'_j are defined by: for every state s_k, $0 \leq k \leq n$, the transition from s_k with input x'_j takes the FSM to state s_j with output y_0.

Let us suppose that X' is a subset of X. By construction, we have that X' distinguishes s_j from the other states of $M_{A,\mathcal{A}}$ if and only if there exists $x_i \in X'$ such that $a_i \in A_j$. Thus, X' is a W-set for $M_{A,\mathcal{A}}$ if and only if $\{a_i | x_i \in X'\}$ is a hitting set.

We now know that any algorithm that solves the MinLength W-set problem for $M_{A,\mathcal{A}}$ and K has also solved the Hitting Set Problem defined by A, $\mathcal{A}$, and K. In addition, $M_{A,\mathcal{A}}$ can be constructed in polynomial time. The result therefore follows from the Hitting Set Problem being NP-hard.

The problem is also in NP. The following result is easier to prove because we know that an FSM has a characterising set of length at most $(n-1)^2$.

Proposition 3. *The MinLength W-set problem is in NP.*

Proof. First observe that a deterministic FSM with n states has a W-set if and only if it has a W-set with no more than $n-1$ input sequences, each of length at most $n-1$. Thus, we can place an upper bound of $(n-1)^2$ on the value of K considered.

A non-deterministic Turing Machine can simply guess a set $\mathcal{W} = \{\bar{x}_1, \ldots, \bar{x}_k\}$ of input sequences of total length at most K. It is then sufficient to check whether this set $\mathcal{W}$ of input sequences is a W-set. For every pair s, s' of distinct states of M, the algorithm proceeds through up to k iterations. Here, in the ith iteration we compute $M(s, \bar{x}_i)$ and $M(s', \bar{x}_i)$. If $M(s, \bar{x}_i) \neq M(s', \bar{x}_i)$ then the Turing Machine records that s and s' are distinguished by $\mathcal{W}$ and moves on to check the next pair of states. If all pairs of states have been considered, and found to be distinguished by $\mathcal{W}$ then the Turing Machine terminates with success. If $M(s, \bar{x}_i) = M(s', \bar{x}_i)$ and $i < k$ then we move to iteration $i+1$ and otherwise ($M(s, \bar{x}_i) = M(s', \bar{x}_i)$ and $i = k$) the Turing Machine concludes that $\mathcal{W}$ is not a W-set.

The outer loop of this algorithm iterates at most $n(n-1)$ times. In addition, the inner loop iterates at most $|\mathcal{W}| \leq (n-1)^2$. The number of iterations is thus of $O(n^4)$ times. Further, given input sequence $\bar{x}_i$ and state s it is possible to compute $M(s, \bar{x}_i)$ in polynomial time.

To conclude, this algorithm takes polynomial time and returns success if and only if $\mathcal{W}$ is a W-set (it succeeds in separating all pairs of states). Thus, a non-deterministic Turing Machine can solve the problem in polynomial time and so the result follows.

From the above, we have the following result.

Theorem 2. *The MinLength W-set problem is* NP-complete*.*

4. Algorithm for Constructing W-sets.

This section introduces a novel algorithm that takes a deterministic minimal FSM and tries to generate a W-set with relatively few elements (please see Algorithm 1 for details). It does so without specifying a bound (K) on the number of elements.

The algorithm receives an FSM M and performs a breadth-first search (BFS), on input sequences, and checks for solutions using a greedy heuristic. The BFS continues until one of the following termination conditions holds.

1. A W-set is formed; or
2. An upper bound on the depth of the tree is exceeded. We set this bound as $n-1$ since, for an FSM with n states, every pair of states can be separated by an input sequence of length at most $n-1$ [26].

Upon receiving its input, the algorithm initiates a *pair set* Δ (Line 1 of Algorithm 1). The pair set (Δ) initially contains the set of all pairs of distinct states (i.e., the set defined by $\binom{S}{2}$) and is used to check which pairs have been separated so far. The algorithm uses Δ to keep the remaining pairs to be separated and hence the algorithm terminates with success if $|\Delta| = \emptyset$. The algorithm applies the BFS iteratively, constructing a tree structure that we call *a BFS tree*. A BFS tree contains a set of vertices V, where each vertex $v \in V$ has four pieces of information: a set of *current states* v_c; a set of *initial states* v_I; an input sequence $v(\bar{x})$; and finally an output sequence $v(\bar{y})$. These are related by: for all $s \in v_I$, there exists a corresponding $s' \in v_C$ such that $(s, \bar{x}, \bar{y}, s') \in h^*$.

Algorithm 1: Minimal W-set for M

Input: FSM $M = (S, s_0, X, Y, h)$ such that $|S| > 1$

Output: A W-set for M

begin

- $\Delta \leftarrow \{(s_i, s_j) | s_i, s_j \in S \text{ and } i < j\}$
- $v \leftarrow (S, S, \varepsilon, \varepsilon)$, $push(v, V)$
- $\ell \leftarrow 0$.
- **while** $l \leq n - 1$ **do**
  - $V^* \leftarrow \emptyset$
  - **while** $V \neq \emptyset$ **do**
    - $v \leftarrow pop(V)$
    - **foreach** *input symbols* $x \in X$ **do**
      - Retrieve $P(v_C, x)$.
      - **foreach** $v^y \in P(v_C, x)$ **do**
        - $v_C^y = \{s' \in S | \exists s \in v_C.(s, x, y, s') \in h\}$
        - $v_I^y = \{s \in v_I | \bar{x}/\bar{y} \in L_M(s)\}$
        - $v^y(\bar{x}) = v(\bar{x})x$, $v^y(\bar{y}) = v(\bar{y})y$
        - $push(v^y, V^*)$
  - $V \leftarrow V^*$
  - $\ell \leftarrow \ell + 1$
  - $Seq \leftarrow \langle\rangle$
  - **foreach** $v \in V$ **do**
    - $\chi \leftarrow$ number of pairs removed from set Δ by $v(\bar{x})$
    - $Seq \leftarrow Seq\langle(\chi, v(\bar{x}))\rangle$
  - $Sort(Seq)$
  - **foreach** $\chi \in Seq$ **do**
    - Remove pairs separated by $\bar{x}^\star$ from Δ
    - **if** *A pair has been separated* **then**
      - $\mathcal{W} \leftarrow \mathcal{W} \cup \{\bar{x}^\star\}$
  - **if** $\Delta = \emptyset$ **then**
    - Return $\mathcal{W}$ after prefix removal.

The BFS tree is constructed as follows. In each iteration (considering a given depth), the algorithm processes the set V that contains the vertices that correspond to the current leaves of the tree. Initially, the vertex set V has a single element v such that $v_C = S$, $v_I = S$, and $v(\bar{x}) = v(\bar{y}) = \varepsilon$ (Line 2 of Algorithm 1). In each iteration, the algorithm, while processing a vertex $v \in V$, then separately *applies* all the inputs from $x \in X$ to v_C (Lines 8-14 of Algorithm 1). By considering the output produced by M, in response to x, the state sets of v are *partitioned*; for all y such that $x/y \in M(v_C, x)$, we introduce a new vertex v^y such that $v_C^y = \{s' \in S | \exists s \in v_C.(s, x, y, s') \in h\}$, $v^y(\bar{x}) = v(\bar{x})x$, $v^y(\bar{y}) = v(\bar{y})y$, and we form v_I^y by simply inheriting the corresponding initial states of s. We use $P(v_C, x)$ to denote the set of vertices that are created by applying x to v_C. The newly created vertices are then added to a set V^* (Line 14 of Algorithm 1). When all the vertices of the current level have been processed, the algorithm copies V^* to V (Line 15 of Algorithm 1), increments the level variable (ℓ) by one (Line 16 of Algorithm 1) and initiates a sequence Seq (Line 17 of Algorithm 1).

This is then followed by analyzing the outcome of the BFS step. In order to do this, the algorithm first gathers all the distinct input sequences from the vertices in the set V^* and forms a set $\bar{X}$. Then, for each input sequence $\bar{x} \in \bar{X}$, it counts the number of pairs of states from Δ that are separated by $\bar{x}$ and stores this value in χ (Lines 18-20 of Algorithm 1). Note that the algorithm does not remove a pair of states from Δ at this step but it removes pairs after processing all the vertices in V (see Lines 22-25), it only counts the number of pairs that can be removed by the input sequence under consideration. Then the algorithm associates this value with the input sequence, i.e., $(\bar{x}, \chi)$, and stores this in set Seq. The algorithm then sorts Seq according to the χ values (Line 21 of Algorithm 1).

After this, the algorithm moves to an iterative step in which it removes pairs (through a heuristic step) from Δ (Δ stores the pairs of states not yet separated). The algorithm uses a (greedy) heuristic in which it selects an input sequence ($\bar{x}$), from one of the new vertices, that is associated with one of the largest χ values. That is, it selects an input sequence that separates as many states as possible. If the algorithm can eliminate one or more pairs from Δ with $\bar{x}$, then the algorithm adds $\bar{x}$ to $\mathcal{W}$. Note that $\bar{x}$ will not always be included; it may be that the pairs of states separated by $\bar{x}$ have been separated by input sequences previously added. This process continues until all pairs have been separated (or it runs out of input sequences) (Lines 22-25 of Algorithm 1).

At the end of an iteration, if Δ is empty then the algorithm returns $\mathcal{W}$ after removing all the sequences that are proper prefixes of other sequences (i.e. given sequences $\mathcal{W} = \{x_1x_2, x_1\}$ we drop x_1 and we have $\mathcal{W} = \{x_1x_2\}$) (Lines 26-27 of Algorithm 1). Otherwise the algorithm continues to execute. We now show that if Algorithm 1 terminates with success, then the set $\mathcal{W}$ returned defines a W-set for M.

Proposition 4. *Let us suppose that v is a vertex formed during the BFS and that v^p is the parent vertex of v. If states s and s' are both in $v_I(v^p)$ but only one of s and s' is in $v_I(v)$ then $v(\bar{x})$ is a separating sequence for s and s'.*

Proof. The result follows from the definition of a separating sequence.

Therefore, if a pair (s, s') of states is removed from Δ then the algorithm has computed a separating sequence for s and s'. Since the algorithm terminates when all the pairs have been removed from Δ, we have the following result.

Theorem 3. *Let us suppose that M is a completely specified minimal deterministic FSM. If Algorithm 1, when given M, returns non-empty set $\mathcal{W}$, then $\mathcal{W}$ is a W-set for M.*

In the worst case, the proposed algorithm constructs a BFS tree by applying every input sequence of length up to $n-1$ and so the worst time complexity of the algorithm is exponential. In the next section we report on experiments that evaluated this algorithm, finding that (compared to the existing algorithm [3]) the proposed algorithm reduces the number of test sequences and the number of inputs in the test suite by 37.3% and 36.4% on average respectively and does so within a reasonable time.

We now provide an example to show how the proposed algorithm (Algorithm 1) can be used to construct a $\mathcal{W}$ set when given the FSM M_1 in Figure 1. The algorithm first evaluates all inputs (lines 8-14), leading to there being a V set. Recall that each element of V is defined by a tuple $(v_I, v_C, v(\bar{x}), v(\bar{y}))$.

$$
\begin{aligned}
V = & \{(\{s_1, s_2\}, \{s_2, s_3\}, x_1, y_1), (\{s_3\}, \{s_2\}, x_1, y_3), (\{s_4\}, \{s_2\}, x_1, y_2), \\
& (\{s_1\}, \{s_1\}, x_2, y_3), (\{s_2, s_3\}\{s_4, s_1\}, x_2, y_2), (\{s_4\}, \{s_1\}, x_2, y_1), \\
& (\{s_1\}, \{s_2\}, x_1, y_1), (\{s_2\}, \{s_3\}, x_1, y_2), (\{s_3\}, \{s_2\}, x_3, y_3), (\{s_4\}, \{s_2\}, x_{3,4})\}
\end{aligned}
$$

After this, the set Seq is constructed (lines 18-20) and it should have the following elements: $Seq = \langle(5, x_1), (5, x_2), (6, x_3)\rangle$. This is because the input x_1 can separate all pairs except (s_1, s_2), the input x_2 can separate all pairs except (s_2, s_3), and finally the input x_3 can separate all 6 pairs of states. Then after the sorting step at line 21 of Algorithm 1, the elements of the set Seq will have the following order $Seq = \langle(6, x_3), (5, x_2), (5, x_1)\rangle$. After processing lines 22-25 the algorithm removes all the elements from Δ and returns the characterising set as $\mathcal{W}_1 = \{x_3\}$.

5. Empirical Evaluation

In this section we describe the experiments carried out to evaluate the proposed W-set generation technique and the results of these. We start, in Section 5.1, by describing the existing W-set generation algorithm and also the W-method. Section 5.2 then outlines the research questions addressed by the experiments. In Section 5.3, we describe the experimental subjects (FSMs). In Section 5.4 we then give the measures used in the evaluation, while Section 5.5 describes the results of the experiments. Finally, in Section 5.6 we discuss threats to validity and how they were addressed.

We used an Intel I7 CPU with 32GB RAM to carry out the experiments. We implemented the W-set construction algorithm (from now on EA) as given in [26], the W-method as given in [3] and the proposed algorithm using C++ on Microsoft Visual studio .Net 2013.

5.1. Existing W-set generation algorithm and the W-method

The existing W-set generation algorithm, given in [26], has two steps.

In the first step, the algorithm determines the set of pairs of states ($\mathcal{S}$) that can be separated by a single input symbol. While doing this the algorithm associates a pair of states in $\mathcal{S}$ with the input sequence (which has only one input) that separates them.

In the second phase, the algorithm enters a loop that ends when all pairs are distinguished. At each iteration of the loop, the algorithm finds pairs of the form (s, s') such that s and s' have not yet been separated and there is some input x that takes (s, s') to a pair that has already been separated by some input sequence $\bar{x}$. When the algorithm finds such a pair (s, s'), it associates (s, s') with the input sequence $x\bar{x}$. The original algorithm given in [26] uses a set of tables (called P_k tables). In our implementation, we used lists instead of tables. Please see Algorithm 2 for more details.

Algorithm 2: The W-Set generation algorithm.

Input: FSM $M = (S, s_0, X, Y, h)$ such that $|S| > 1$

Output: A W-set $\mathcal{W}$ for M

begin

1 $\mathcal{S} \leftarrow \{\}$

2 **foreach** $(s, s') \in S,\ s \neq s'$ **do**

3 **if** *there exist* x, y, y' *such that* $x/y \in L_M(s),\ x/y' \in L_M(s')$ *and* $y \neq y'$ *(randomly pick one such* $x \in X$*)* **then**

4 $\mathcal{S} \leftarrow \mathcal{S} \cup \{(s, s'), x\}$

5 **while** *Not all pairs of states of* M *are separated* **do**

6 $\mathcal{S}' \leftarrow \{\}$

7 **foreach** *pair of state* (s, s') *that is not included in* $\mathcal{S}$ **do**

8 **if** *there exist* $x, y, \bar{x}, s_1, s'_1$ *such that* $(s, x, y, s_1) \in h,\ (s', x, y, s'_1) \in h$ *and* $(s_1, s'_1, \bar{x}) \in \mathcal{S}$ **then**

9 $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{(s, s'), x\bar{x}\}$

10 $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}'$

11 **foreach** $\{(s, s'), \bar{x}\} \in \mathcal{S}$ **do**

12 $\mathcal{W} \leftarrow \mathcal{W} \cup \bar{x}$

13 Return $\mathcal{W}$ after prefix removal.

As a running example consider the FSM given in Figure 1. After initialization, lines 2-4 of Algorithm 2 are executed and this may lead to the following set.

$$\mathcal{S} = \{\{(s_1, s_2), x_2\}, \{(s_1, s_3), x_1\}, \{(s_1, s_4), x_1\}, \{(s_2, s_3), x_1\}, \{(s_2, s_4), x_1\}, \{(s_3, s_4), x_1\}\}$$

Since all the pairs are separated, the algorithm executes lines 11-13 and returns $\mathcal{W}_2 = \{x_1, x_2\}$.

Note that the existing algorithm could also return $\{x_3\}$ as the W-set. However we show in the experiments section that this need not be the case.

We now describe the W-method test generation algorithm, which has three phases to construct a test suite (TS):

$$TS = r.R.\mathcal{W} \cup r.R.X.\mathcal{W}$$

where r denotes the reset operation that brings the FSM to the initial state, and R denotes the set of prefix closed sequences that reaches each state from the initial state. The details of the W-method are given in Algorithm 3.

In lines 1-3 of Algorithm 3, the algorithm constructs the R sequences, (ε is used to reach the initial state s_0). Then in lines 4-6 of Algorithm 3, it introduces a new set Z by concatenating the elements of $\{\{\epsilon\} \cup X\}$ with $\mathcal{W}$. That is it forms

$$Z = \{\{\epsilon\} \cup X\} \cdot \mathcal{W}$$

In lines 7-9 of Algorithm 3, the W-method constructs the test suite by concatenating the elements of R and Z. i.e., the test suite is $R \cdot Z$. The W-method returns a test suite by removing input sequences that are proper prefixes of other sequences.

The W-method has a parameter m, which is an upper bound on the number of states of the system under test; in all cases, we set m to be equal to the number of states of the specification FSM. The time complexity of the W-method given in [3] is $O(n^3 * p)$ where $p = |X|$, when $m = n$. The source code for the implementations can be found through `https://zenodo.org/badge/latestdoi/258352564`.

We now describe the application of the W-method using the previously mentioned characterizing sets ($\mathcal{W}_1 = \{x_3\}$ and $\mathcal{W}_2 = \{x_1, x_2\}$) and FSM M_1. First, consider the case where Algorithm 3 is given FSM M_1 (Figure 1) and $\mathcal{W}_1$ as inputs. The algorithm will construct the R and the Z sets as follows:

$$R = \{\varepsilon, x_1, x_1x_1, x_1x_2\}$$

$$Z = \{\varepsilon, x_1, x_2, x_3, x_1x_1, x_1x_2, x_1x_3, x_1x_1x_1, x_1x_1x_2, x_1x_1x_3, x_1x_2x_1, x_1x_2x_2, x_1x_2x_3\}$$

Since $\mathcal{W}_1 = \{x_3\}$, after prefix removal the test suite TS_1 is as follows

$$\begin{aligned} TS_1 = \{ & x_2x_3, x_3x_3, x_1x_3x_3, x_1x_1x_1x_3, x_1x_1x_2x_3, x_1x_1x_3x_3, \\ & x_1x_2x_1x_3, x_1x_2x_2x_3, x_1x_2x_3x_3\} \end{aligned}$$

Therefore TS_1 contains 9 sequences with 31 inputs.

Now suppose that Algorithm 3 is given M_1 and $\mathcal{W}_2$ as inputs. The algorithm will construct the same R and Z sets. However, because $\mathcal{W}_2 = \{x_1, x_2\}$, after prefix removal, the test suite set TS_2 is as follows.

$$\begin{aligned} TS_2 =\{&x_2x_1, x_3x_1, x_2x_2, x_3x_2, x_1x_3x_1, x_1x_3x_2, x_1x_1x_1x_1,\\ &x_1x_1x_2x_1, x_1x_1x_3x_1, x_1x_1x_1x_2, x_1x_1x_2x_2, x_1x_1x_3x_2,\\ &x_1x_2x_1x_1, x_1x_2x_2x_1, x_1x_2x_3x_1, x_1x_2x_1x_2, x_1x_2x_2x_2, x_1x_2x_3x_2\}\end{aligned}$$

The test suite TS_2 contains 18 sequences with 62 inputs. So in this example, the proposed method reduced both the number of inputs and the number of sequences by 50%.

Algorithm 3: The W-Method.

Input: FSM $M = (S, s_0, X, Y, h)$, and W-set $\mathcal{W}$ for M
Output: A test suite TS for M

```
begin
    Initialize R ← {ε}, Z ← ∅, and TS ← ∅
    foreach s ∈ S \ {s0} do
        Find an input sequence x̄ that takes M from s0 to s and add this sequence to R
          (R ∪ {x̄})
    foreach x̄ ∈ R do
        foreach x ∈ X ∪ {ϵ} do
            Z∪{x̄x}
    foreach x̄ ∈ Z do
        foreach x̄′ ∈ W do
            TS ← TS ∪ {x̄x̄′}
    Return TS after prefix removal.
```

5.2. Research questions

The experiments were designed to address the following research questions.

RQ1 Does the proposed algorithm tend to return W-sets with fewer input sequences than the standard W-set generation algorithm?

RQ2 Does the proposed algorithm tend to lead to test suites with fewer input sequences, when the W-set is used within the W-method?

RQ3 Does the proposed algorithm tend to lead to test suites with fewer inputs in total (sum of lengths of input sequences), when the W-set is used within the W-method?

RQ4 Does the proposed algorithm run in a reasonable amount of time?

Observe that the first research question directly relates to the objective of the proposed algorithm: to return a W-set with relatively few input sequences. The second research questions is related to the motivation for producing such a W-set, which is that its use should lead to test suites with a relatively small number of input sequences (and so, also, resets). Thus, positive answers to the first two research questions would suggest that the algorithm is effective in returning small W-sets and its use leads to relatively small test suites. We considered the third research question in order to check that any benefits of having smaller test suites (fewer input sequences) are not potentially outweighed by the overall test suite length (the sum of the lengths of the input sequences). The final research question addresses the scalability of the proposed algorithm.

5.3. Experimental subjects

In order to compare the proposed algorithm and the existing W-set generation algorithm, we randomly generated two classes of FSMs and we also used benchmark FSMs. In this section, we provide more details regarding the FSMs used.

5.3.1. FSMs in Class I

The FSMs in the first class (C1) were generated as follows. First, for each input x and state s we randomly assigned the next state and output values. After an FSM M was generated we checked its suitability as follows. We checked whether M was initially connected and was also minimal (and so has a W-set). If the FSM passed these tests then we included it into C1, otherwise we omitted this FSM and produced another one. Consequently, all generated FSMs were initially connected, minimal, and had W-sets.

By using this procedure, we constructed six sets of 1000 FSMs with n states, for each $n \in \{50, 60, \ldots, 150\}$. In all cases, there were three input symbols and three output symbols. In total, we therefore constructed $11,000$ FSMs for the first class of FSMs.

5.3.2. FSMs in Class II

Note that for the FSMs in C1, the next state of each transition is randomly selected, hence the in-degrees[3] of different states will be similar. In order to reduce the potential impact of this, we also generated a second class (C2), where we aimed to have FSMs with less similar in-degree values. To create such an FSM, we first randomly generated an FSM as before and then selected a subset $\bar{S}$ of states, the intention being to give these states higher in-degree values than the states in $S \setminus \bar{S}$. To create higher in-degree values for the states in $\bar{S}$, we randomly selected a subset Γ of the set of transitions (where each element of Γ is a pair (s, x) denoting the transition from state s with input symbol x). We then forced the transitions in Γ to end in states in $\bar{S}$. Similar to C1, after an FSM M was generated we checked its suitability.

When constructing this set of FSMs it was necessary to choose the cardinalities of $\bar{S}$ and Γ. If $|\Gamma|$ was too large and $|\bar{S}|$ was too small then one might not be able to construct a connected FSM, or one might tend to construct FSMs that are not minimal and so do not have a W-set. On the other hand, if $|\Gamma|$ was too small and $|\bar{S}|$ was too large then the in-degrees of the states will again be similar.

In these experiments we chose $|\bar{S}|$ to be 10% of the states and we set $|\Gamma|$ to be 30% of the transitions. We observed that if $|\Gamma|$ is a much higher proportion of the number of transitions then it takes too much time to construct suitable FSMs.

[3]The in-degree of a state s is the number of transitions ending in s.

We constructed 11,000 FSMs in $C2$, with the number of states n being in $\{50, 60, \ldots, 150\}$; for each n there were 1,000 FSMs. We again used input and output alphabets of size three.

5.3.3. Benchmark FSMs

In addition to the randomly generated FSMs, we used benchmark FSM specifications from a repository[4]. The FSM specifications were available in the *Dot* format. In order to process the FSMs, we converted the *Dot* file format to our FSM specification format. We used 16 FSMs from the benchmark; these were minimal, deterministic, had a W-set and had fewer than 10 input bits[5]. We can split the set of benchmark FSMs used into groups. The first group originated from the ACM/SIGDA benchmarks, a set of test suites (FSMs) used in workshops between 1989 and 1993 [43] These FSMs, ranging from simple circuits to circuits obtained from industry, were: *DVRAM*, *Ex4*[6], *Log*, *Rie*, and *Shift Register*. For these FSMs we flattened the transitions with don't care ("-") symbols. That is, for each string with don't care symbol, we introduce two strings with 0 and 1 symbols, i.e., -01010 will be replaced to lead to 101010 and 001010 We also used FSMs that model the alternating bit protocol and FSMs obtained by learning algorithms applied to Coffee Machine and Maestro credit card case studies [44]. Table 2 gives the number of states and transitions of the benchmark FSMs.

5.4. Measures used

The experiments applied the proposed algorithm and the previously published algorithm, using these to generate W-sets and then test suites (using the W-method). Thus, for each algorithm and group of FSMs, such as all FSMs

[4] The repository can be accessed via `https://automata.cs.ru.nl/`.

[5] This was for practical reasons since the circuits receive inputs in bits and b bits correspond to 2^b inputs.

[6] FSM specification Ex4 is partially specified. We complete the missing transitions by adding self looping transitions with a special output symbol, and do not use these inputs for W-set construction.

Name	No of states	No of transitions
Shift Register	8	16
Coffee Machine	6	24
ABPSender.flat01	11	55
Maestro	6	84
ABPSender.flat02	15	105
ABPChannelflame.flat02	10	110
ABPSender.flat03	19	171
ABPSender.flat04	23	253
ABPChannel.Frame.flat03	17	306
ABPSender.flat05	27	351
ABPSender.flat06	31	465
ABPSender.flat07	35	595
Ex4	14	896
Log	17	8704
DVRAM	35	8960
Rie	29	14848

Table 2: Benchmark FSMs and their sizes

in C1 with a given number of states, we computed the following: mean W-set size (number of input sequences); the mean test suite size (number of input sequences); and the mean test suite length (total number of inputs).

In the analysis, we also used three measures to compare the results produced by the two W-set generation techniques. These measures are directly associated with the first three research questions. The first measure ($M1$) is the percentage reduction in the number of elements of the W-sets constructed using the proposed algorithm (P), when compared to the W-sets returned by the existing algorithm (EA). If we let $\mathcal{W}_{EA}$ be the W-set returned by EA, and $\mathcal{W}_P$ be the

W-set returned by P, then this measure can be defined as follows.

$$M1 = \frac{|\mathcal{W}_{EA}| - |\mathcal{W}_P|}{|\mathcal{W}_{EA}|} * 100$$

The second measure ($M2$) gives the percentage reduction in the number of test sequences returned by the W-method when it is fed with the W-set returned by the proposed algorithm. Let $T(.)$ be the number of test sequences given by the parameter. We will use $EA(M)$ to denote the test suite returned by the W-method using $\mathcal{W}_{EA}$ and $P(M)$ to denote the test suite returned by the W-method when using $\mathcal{W}_P$. Then $M2$ is computed as follows:

$$M2 = \frac{T(EA(M)) - T(P(M))}{T(EA(M))} * 100$$

The final measure ($M3$) is the percentage reduction in the total number of inputs in the test suites returned by the W-Method when using $\mathcal{W}_P$ and $\mathcal{W}_{EA}$ respectively. $M3$ is computed as follows:

$$M3 = \frac{Lt(EA(M)) - Lt(P(M))}{Lt(EA(M))} * 100$$

where $Lt(.)$ returns the sum of the lengths (number of inputs) of the input sequences in the test suite given in the parameter.

5.5. Results and Evaluation

In order to evaluate the relative performance of different approaches, for each FSM M, we separately computed W-sets using the proposed algorithm and the existing algorithm. We then generated test suites using the W-method and computed the values of the three measures. In these experiments, we used two existing tools to check all of the W-sets and test suites produced during the experiments. Given an FSM M, one tool [37] checks that a given set of input sequences is a W-set for M and the other tool [45] checks whether a given set of input sequences defines a checking experiment for M.

We start with the results for the first set of experimental subjects, the randomly generated FSMs in C1. The results are given in Figure 2.

We observe from Figure 2b that, for the proposed algorithm, the number of input sequences in the W-sets increases only very slowly as the number of states increases. In contrast, the number of input sequences increases quite noticeably when we use the traditional approach. Recall that the proposed algorithm selects the input sequence that distinguishes more pairs than the others while constructing the W-set (Lines 21-25 of Algorithm 1). Therefore, as the number of states increases the algorithm has more options for reducing the number of input sequences and hence can pick input sequences that distinguish more pairs than others. This helps explain why the W-set size grows more slowly when using the proposed algorithm.

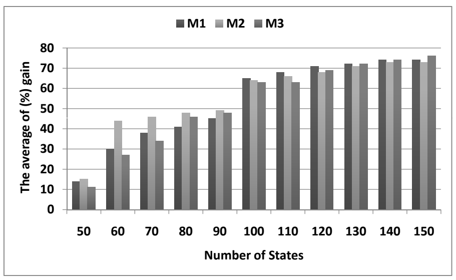
The results, regarding the size and length of the test suites returned by the W-method, are given in Figure 2c and Figure 2d respectively. In both figures, we observe that using the P algorithm leads to test suites with fewer elements and fewer inputs than the EA algorithm.

Finally, the values of the metrics can be found in Figure 2a. As would be expected, given the other results, the percentage gains increase as the number of state increase, reaching nearly 80% for FSMs with 150 states.

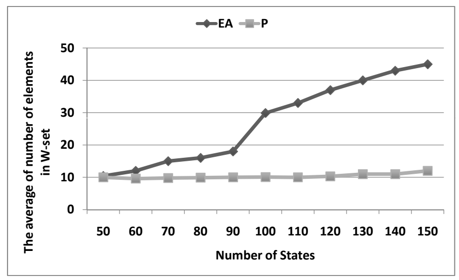
The results for C2 are given in Figure 3. We observe from Figure 3b that the sizes of the W-sets plateaued for the FSMs if the proposed algorithm is used. In contrast, the mean W-set size increases steadily with the number of states when computed by EA. This again stems from the heuristic step taken by P. The results regarding test suite size and length are given in Figures 3c and 3d respectively. Similar to before, the proposed algorithm leads to smaller and shorter test suites, with the differences increasing as the number of states increases.

The values of the three metrics are given in Figure 3a. The results are similar to those with C1: there are savings associated with all three metrics and these savings increase as the number of states increases.

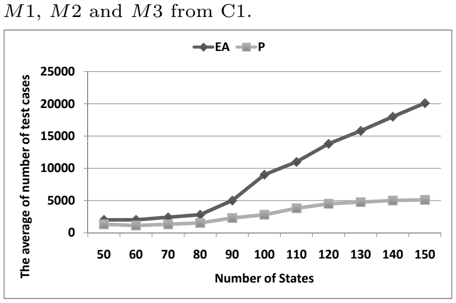
To investigate the results further, we conducted a non-parametric two-tailed paired hypothesis test on the results where the null hypothesis stated that the paired $P(M), EA(M)$ values were equal and the significance level was $\alpha =$



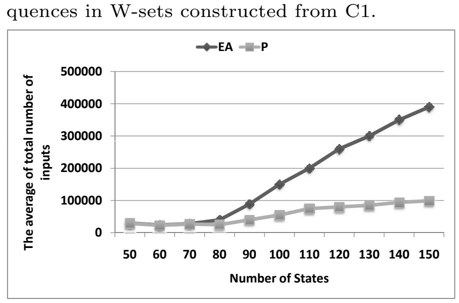
(a) Averages of the gain with respect to metrics $M1$, $M2$ and $M3$ from C1.



(b) Averages of the total number of input sequences in W-sets constructed from C1.

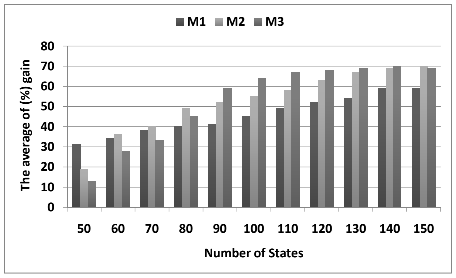


(c) Averages of the total number of input sequences in test suites constructed from C1.

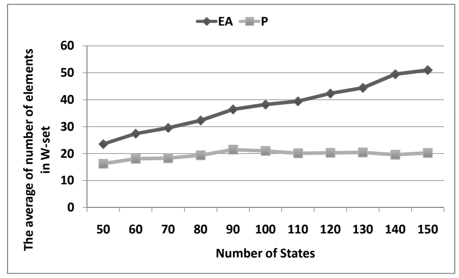


(d) Averages of the total number of inputs in test suites constructed from C1.

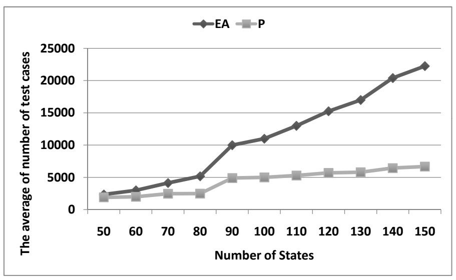
Figure 2: Performance comparison of algorithms $\mathcal{W}_{EA}$ and $\mathcal{W}_P$ on different metrics observed from C1.



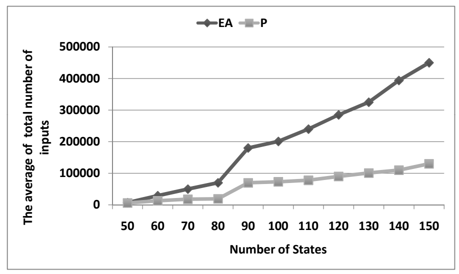
(a) Averages of the gains with respect to metrics $M1$, $M2$ and $M3$ from C2.



(b) Averages of the total number of input sequences in W-sets constructed from C2.

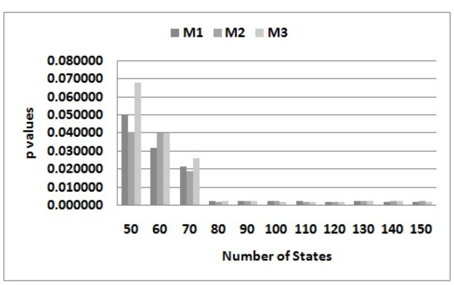


(c) Averages of the total number of input sequences in test suites constructed from C2.

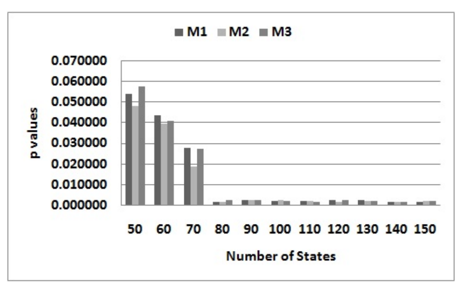


(d) Averages of the total number of inputs in test suites constructed from C2.

Figure 3: Performance comparison of algorithms $\mathcal{W}_{EA}$ and $\mathcal{W}_{P}$ on different metrics observed from C2.

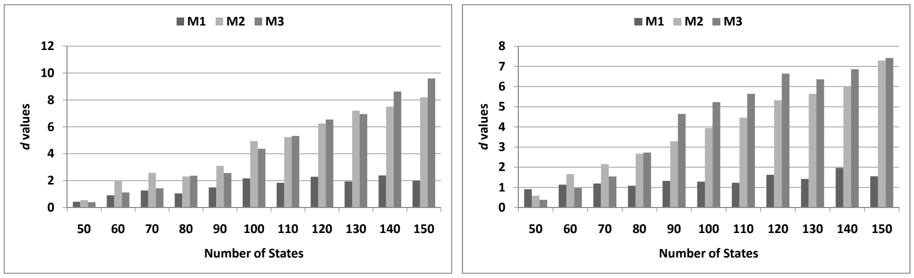


(a) p-values for C1



(b) p-values for C2

Figure 4: p-values obtained from non-parametric paired hypothesis test (Wilcoxon-test) for the experiments conducted on $C1$ and $C2$ where $\alpha = 0.05$.



(a) Cohen's d metric results on results of metrics $M1$, $M2$ and $M3$ retrieved from C1.

(b) Cohen's d metric results on results of metrics $M1$, $M2$ and $M3$ retrieved from C2.

Figure 5: Cohen's d metric results on test suites C1 and C2.

0.05 [46]. We used the R tool to conduct the statistical evaluation [47]. The resultant p-values are given in Figure 4. We can see that when $n \geq 60$, the p value is less than 0.05 for the $M1$, $M2$, and $M3$ values for both $C1$ and $C2$.

Moreover, we complemented our statistical analysis by considering the statistical effect size through computing Cohen's distance d for $M1$, $M2$, and $M3$ metrics computed over the FSMs in C1 and C2 (Figure 5) [48]. As can be seen from Figure 5a and Figure 5b, in all cases the effect size were larger than 0.5 and so the effect size was large.

Considering the results provided for C1 and C2 we can deduce that there is clear evidence that the proposed algorithm leads to smaller W-sets which addresses the *first research question*.

The results also provide information about the mean test suite sizes and here we see an average reduction of 37.3% in the number of test sequences. In addition, the reduction increases as the number of states increases. Moreover, the effect of non-uniform transition distribution (C2) was found to be negligible. We investigated the effect of non-uniform transition distribution using the one way ANOVA test for which the null hypothesis (the mean value is the same for C1 and C2) was accepted for measures $M1$, $M2$, and $M3$ [47].

Likewise, Figure 4 and Figure 5 indicate that when $n \geq 60$, the p value is less than 0.05 for $M2$, with both $C1$ and $C2$, and with large effect size (Figure 5). This addresses the *second research question*, and suggests that the proposed

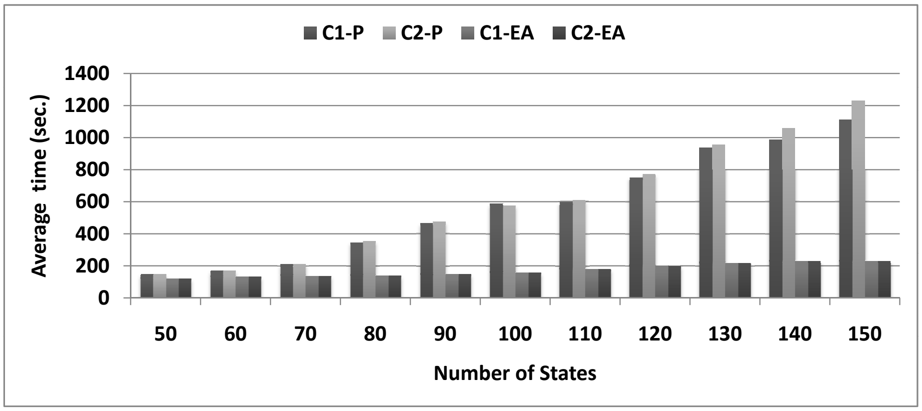


Figure 6: Time comparison of the W-set generation algorithms. X axis labels the number of states, and Y axis labels the time (seconds).

algorithm tends to lead to smaller test suites.

In addition to the above, we observe a reduction of 36.4% on average in the total number of inputs in a test suite. Similar to before, the reduction increases as the number of states increases. Again, from Figure 4 and Figure 5, we can see that when $n \geq 60$, the p value is less than 0.05 for $M3$ with both $C1$ and $C2$. The effect sizes are also large (Figure 5). This addresses the third research question.

We recorded the time taken, by the two methods, to construct the W-sets, with the results being given in Figure 6. The proposed algorithm was found to be slower; 2.218 times slower on average. This generally stems from the fact that the proposed method generates and checks a high number of vertices, i.e. the number of vertices grows with $l^{|X|}$. However, the times were acceptable for even the larger FSMs. Note that, for a given FSM M, the developer will run the proposed algorithm once, that is, it is a one-time computation. In contrast, the resultant test suite will typically be run many times (for example, in regression testing) and so it makes sense to "invest" in the generation of a smaller test suite. In addition, test suite execution time will often exceed test suite generation time.

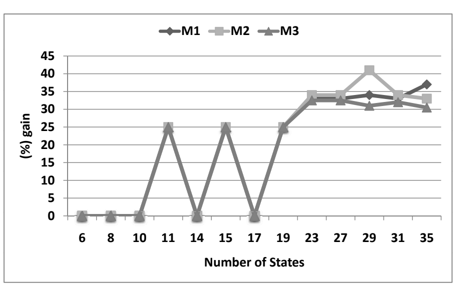
The results of the experiments conducted on the benchmarks are given in Table 3. We see that, for some of the specifications, there is no difference in the number of elements in the W-sets and the number of test sequences. However, this is not too surprising since these FSMs are relatively small. Nevertheless, for 9 out of 16 FSMs the results are promising: we observe that the proposed algorithm reduces the number of elements in the W-set, reduces the number of test sequences, and reduces the total number of inputs of the test suite as well.

Figure 7a shows the values of the metrics, M1-M3, plotted against the number of states of the benchmark FSMs. These results indicate that the FSMs where there were no benefits were FSMs with fewer states, suggesting that the reduction tends to increase with the number of states. We also plotted the values of the metrics against the number of transitions of the FSMs (Figure 7b)). Interestingly, there is no clear pattern. This suggests that the savings introduced by the proposed algorithm depend more on the number of states than the number of transitions.

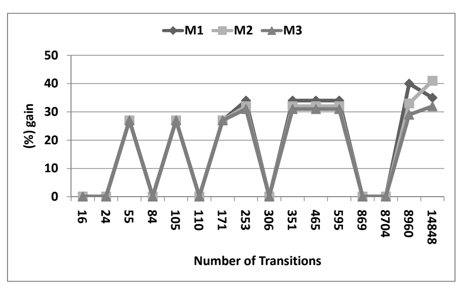
Table 3 also shows the time taken by the two algorithms. We see that the time required to generate the W-sets is, in general, higher when the proposed algorithm is used. However, the time is negligible for these examples.

5.6. Threats to validity

In this section we discuss some potential threats to the validity of the experimental results and how these were addressed.



(a) Percentage gain-number of states chart for the benchmark FSMs with respect to M1, M2 and M3 metrics.



(b) Percentage gain-number of transitions chart for the benchmark FSMs with respect to M1, M2 and M3 metrics.

Name	M1	M2	M3	P(msec.)	EA(msec.)
Shift Register	0	0	0	25	24
Coffee Machine	0	0	0	< 1	< 1
ABPSender.flat01	25%	25%	25%	1	1
Maestro	0	0	0	< 1	< 1
ABPSender.flat02	25%	25%	25%	1	1
ABPChannelflame.flat_02	0	0	0	1	1
ABPSender.flat03	25%	25%	25%	1	1
ABPSender.flat04	33%	34%	32%	3	2
ABPChannelflame.flat03	0	0	0	1	1
ABPSender.flat05	33%	34%	32%	4	3
ABPSender.flat06	33%	34%	32%	6	7
ABPSender.flat07	33%	34%	32%	15	10
Ex4	0	0	0	270	195
Log	0	0	0	260	234
DVRAM	40%	32%	29%	567	267
Rie	34%	41%	31%	689	304

Table 3: Results of experiments conducted on benchmark FSMs

First, there are threats to generalisability. We evaluated the performance of our algorithm by using randomly generated FSMs. It is possible that the performance of our proposed algorithm differs for FSMs used in real-life situations. Although using random FSMs is normal in this field, in order to evaluate the generalization of the proposed algorithm, we also evaluated the proposed algorithm using case studies consisting of benchmark FSM specifications (Section 5.3.3). We see that the experimental results obtained from randomly generated FSMs are similar to the results obtained with the larger benchmark FSMs. However, for small FSMs, we did not observe any difference between the proposed algorithm and the existing algorithm.

There are also threats to internal validity and the possibility that one or more of the implementations were incorrect. In order to reduce this threat, we also used two existing tools [37, 45]. The first tool checks if a given set of sequences is a W-set for an FSM. The second tool checks whether or not a given set of input sequences defines a checking experiment for an FSM M.

6. Related Work

Minimizing the W-set is inherently related to i) minimizing the size of separating sequences and ii) test suite minimization.

To begin with, to our knowledge, this work is the first to investigate the problem of finding a small W-set. In [49] the author discusses an algorithm that constructs W-sets with short sequences. Note that the algorithm presented in this previous work is identical to the algorithm given in [26].

However, for a given FSM there are a number of different types of sequences that can separate states, with examples including *adaptive distinguishing sequences* (ADSs), and *unique input output sequences (UIOs).* Previous work explored the problem of construct minimal ADSs. It was proven that minimizing the height of an ADS (in fact minimizing ADS size with respect to some other metrics as well) is an NP-hard problem [50]. Türker and Yenigün proposed two heuristics as a modification of the existing ADS generation algorithm for minimization [50]. Recently Türker et al. also presented a BFS based algorithm called the *lookahead based algorithm* (LA) for minimizing ADSs [51]. In LA, a branch of a BFS tree is extended if it satisfies certain conditions. However, the algorithm proposed in this paper expands each branch of a BFS tree until the W-set is constructed. Regarding UIOs, the literature contains several works on reducing the size of these sequences. First, the algorithm introduced in [9] is a brute-force algorithm based on a BFS tree. Therefore, this algorithm finds the shortest such sequences. Later in [52], the author introduced a heuristic to find UIOs faster.

The proposed W-set generation algorithm includes a step in which the choice of input sequence is based on how many pairs of states are separated by the sequences. The approach taken is a form of greedy algorithm, in which the test sequence with greatest 'score' (number of pairs of states separated) is chosen. Greedy algorithms have been used in a number of areas of software testing, including two topics relevant to regression testing: test minimization and (later) test prioritization (see, for example, [53, 54, 55]). The context of this previous work is that we have a regression test suite T and wish to either reduce the cost of applying regression testing by only using a subset of T (test minimization) or to apply the test cases in an order that means that any failures are likely to be observed relatively early.

One possibility would have been to simply sort the sequences based on how many pairs of states are distinguished and this would then have been similar to the Greedy Algorithm that was devised for prioritizing test cases based on coverage information (see, for example, [55]). This prioritization technique involves computing the coverage provided by each test case and then simply sorting the test cases based on the coverage scores. However, instead, whenever the sequence length is increased we use an updated set Δ of pairs of states not yet covered and the choice of next input sequence is based on the updated Δ. This is a little like the so-called Additional Greedy approach to test prioritisation. Here, in each iteration, the approach chooses the test case providing highest additional coverage and then updates the coverage information [55].

Although Additional Greedy has been found to be an effective prioritization technique, it has been observed that it can be sub-optimal. As a result, a number of alternatives have been explored. In particular, it has been found that metaheuristic algorithms, such as Genetic Algorithms, can outperform Additional Greedy (see, for example, [56, 57]). It may well be that the proposed algorithm can be improved by incorporating such algorithms. However, such approaches are likely to significantly increase the (algorithm) execution time and so probably will only be of value in situations in which the benefits of a small reduction in test suite size are not outweighed by such an increase in W-set generation time.

7. Conclusion

Software testing is typically performed manually and is an expensive, error prone process. This has led to interest in automated test generation, including significant interest in model based testing (MBT). Most MBT techniques generate test suites from either finite state machines (FSMs) or labelled transition systems, with such a model potentially representing the semantics of a specification in a richer language.

In this paper, we investigated the problem of computing a minimal W-set for a given deterministic, minimal completely specified FSM. We introduced the associated decision problem and showed that the problem of deciding whether an FSM has a W-set with at most K input sequences is PSPACE-complete. In contrast, the problem of deciding whether an FSM has a W-set with total length K is NP-complete.

The initial motivation for minimizing a W-set was the use of W-sets in the context of test suite generation. Ideally one uses a minimal W-set, since the W-set is used in state recognition and state verification. Therefore, the size of a test suite generated using a W-set should correlate with the number of elements of the W-set. Due to the hardness of W-set minimization, a heuristic algorithm was proposed. Experiments were conducted to evaluate the proposed algorithm.

In these, the use of a W-set returned by the proposed algorithm reduced the number of test sequences in a test suite by 37.3% on average, with the total number of inputs being reduced by 36.4% on average.

Although the proposed algorithm was evaluated in the context of the W-method, there are potential implications for some other FSM-based test generation techniques. The Wp [9], the HSI-method [28], the HIS [58][7], H [35, 32], and incremental methods (such as [59]) rely on the presence of a W-set and one might conjecture that the use of a small W-set is likely to lead to these returning more efficient test suites.

There are a number of additional lines of future work. First, it would be interesting to explore realistic conditions under which the decision and optimisation problems can be solved in polynomial time. Such conditions might lead to new notions of testability. Although the results of the experiments suggest that the use of relatively small W-sets leads to test suites that require fewer resets, it would be interesting to extend the experiments and possibly also consider the Wp, HIS and SPY algorithms [9, 58, 22]. Finally, it would be interesting to investigate complexity results and effective algorithms that can generate minimum W-sets from non-deterministic FSMs.

Acknowledgement

This work is dedicated to the memory of Prof. Hasan Ural, who was our mentor/friend. Prof. Ural was a good teacher who introduced new scientists to the testing community, he was also a dedicated scientist who provided important contributions relentlessly to the testing spectra for nearly half a century.

[7]Note, we use the naming convention used in [32] and use the HIS method to refer to the test suite generation algorithm.

References

[1] A. Friedman, P. Menon, Fault detection in digital circuits, Computer Applications in Electrical Engineering Series, Prentice-Hall, 1971.

[2] A. Aho, R. Sethi, J. Ullman, Compilers, principles, techniques, and tools, Addison-Wesley series in computer science, Addison-Wesley Pub. Co., 1986.

[3] T. S. Chow, Testing software design modelled by finite state machines, IEEE Transactions on Software Engineering 4 (1978) 178–187.

[4] E. Brinksma, A theory for the derivation of tests, in: Proceedings of Protocol Specification, Testing, and Verification VIII, North-Holland, Atlantic City, 1988, pp. 63–74.

[5] A. Dahbura, K. Sabnani, M. Uyar, Formal methods for generating protocol conformance test sequences, Proceedings of the IEEE 78 (8) (Aug) 1317–1326.

[6] D. Lee, K. Sabnani, D. Kristol, S. Paul, Conformance testing of protocols specified as communicating finite state machines-a guided random walk based approach, IEEE Transactions on Communications 44 (5) (May) 631–640.

[7] D. Lee, M. Yannakakis, Principles and methods of testing finite-state machines - a survey, Proceedings of the IEEE 84 (8) (1996) 1089–1123.

[8] S. Low, Probabilistic conformance testing of protocols with unobservable transitions, in: 1993 International Conference on Network Protocols, Oct, pp. 368–375.

[9] K. Sabnani, A. Dahbura, A protocol test generation procedure, Computer Networks 15 (4) (1988) 285–297.

[10] D. P. Sidhu, T.-K. Leung, Formal methods for protocol testing: A detailed study, IEEE Transactions on Software Engineering 15 (4) (1989) 413–426.

[11] R. V. Binder, Testing Object-Oriented Systems: Models, Patterns, and Tools, Addison-Wesley, 1999.

[12] M. Haydar, A. Petrenko, H. Sahraoui, Formal verification of web applications modeled by communicating automata, in: Formal Techniques for Networked and Distributed Systems FORTE, Vol. 3235 of LNCS, Springer-Verlag, Madrid, 2004, pp. 115–132.

[13] A. Betin-Can, T. Bultan, Verifiable concurrent programming using concurrency controllers, in: Proceedings of the 19th IEEE international conference on Automated software engineering, IEEE Computer Society, 2004, pp. 248–257.

[14] I. Pomeranz, S. M. Reddy, Test generation for multiple state-table faults in finite-state machines, IEEE Transactions on Computers 46 (7) (1997) 783–794.

[15] M. Utting, A. Pretschner, B. Legeard, A taxonomy of model-based testing approaches, Software Testing, Verification and Reliability 22 (5) (2012) 297–312.

[16] W. Grieskamp, N. Kicillof, K. Stobie, V. A. Braberman, Model-based quality assurance of protocol documentation: tools and methodology (2011).

[17] A. V. Aho, A. T. Dahbura, D. Lee, M. U. Uyar, An optimization technique for protocol conformance test generation based on UIO sequences and rural chinese postman tours, in: Protocol Specification, Testing, and Verification VIII, Elsevier (North-Holland), Atlantic City, 1988, pp. 75–86.

[18] F. C. Hennie, Fault-detecting experiments for sequential circuits, in: Proceedings of Fifth Annual Symposium on Switching Circuit Theory and Logical Design, Princeton, New Jersey, 1964, pp. 95–110.

[19] G. Gonenc, A method for the design of fault detection experiments, IEEE Transactions on Computers 19 (1970) 551–558.

[20] S. T. Vuong, W. W. L. Chan, M. R. Ito, The UIOv-method for protocol test sequence generation, in: The 2nd International Workshop on Protocol Test Systems, Berlin, 1989.

[21] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, A. Ghedamsi, Test selection based on finite state models, IEEE Transactions on Software Engineering 17 (6) (1991) 591–603.

[22] A. da Silva Simão, A. Petrenko, N. Yevtushenko, On reducing test length for FSMs with extra states, Software Testing, Verification and Reliability 22 (6) (2012) 435–454.

[23] H. Ural, K. Zhu, Optimal length test sequence generation using distinguishing sequences, IEEE/ACM Transactions on Networking 1 (3) (1993) 358–371.

[24] A. Petrenko, N. Yevtushenko, Testing from partial deterministic FSM specifications, IEEE Transactions on Computers 54 (9) (2005) 1154–1165.

[25] Z. Kohavi, Switching and Finite State Automata Theory, McGraw-Hill, New York, 1978.

[26] A. Gill, Introduction to The Theory of Finite State Machines, McGraw-Hill, New York, 1962.

[27] M. P. Vasilevskii, Failure diagnosis of automata, Cybernetics 4 (1973) 653–665.

[28] G. Luo, A. Petrenko, G. V. Bochmann, Selecting test sequences for partially-specified nondeterministic finite state machines, in: Protocol Test Systems, IFIP The International Federation for Information Processing, Springer US, 1995, pp. 95–110.

[29] D. Angulin, Learning regular sets from queries and counterexamples, Information and Computation 75 (1987) 87–106.

[30] D. Huistra, J. Meijer, J. van de Pol, Adaptive learning for learn-based regression testing, in: Formal Methods for Industrial Critical Systems, Vol. 11119 of Lecture Notes in Computer Science, Springer International Publishing, Cham, 2018, pp. 162–177.

[31] N. Yang, K. Aslam, R. Schiffelers, L. Lensink, D. Hendriks, L. Cleophas, A. Serebrenik, Improving model inference in industry by combining active and passive learning, in: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2019, pp. 253–263.

[32] R. Dorofeeva, K. El-Fakih, N. Yevtushenko, An improved conformance testing method, in: Proceedings of the 25th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems, FORTE'05, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 204–218.

[33] A. Petrenko, G. v. Bochmann, R. Dssouli, Conformance relations and test derivation, in: Proceedings of Protocol Test Systems VI (C-19), 1993, pp. 157–178.

[34] A. D. Friedman, P. R. Menon, Fault detection in digital circuits, Prentice-Hall Englewood Cliffs, N.J, 1971.

[35] I. Koufareva, M. Dorofeeva, A novel modification of W-method, Joint Bull. Novosibirsk Comput (2002) 69–81.

[36] E. P. Hsieh, Checking experiments for sequential machines, IEEE Transactions on Computers 20 (1971) 1152–1166.

[37] R. M. Hierons, U. C. Türker, Parallel algorithms for generating harmonised state identifiers and characterising sets, IEEE Transactions on Computers 65 (11) (2016) 3370–3383.

[38] R. M. Hierons, Minimizing the number of resets when testing from a finite state machine, Information Processing Letters 90 (6) (2004) 287–292.

[39] R. Dorofeeva, K. El-Fakih, S. Maag, A. R. Cavalli, N. Yevtushenko, FSM-based conformance testing methods: a survey annotated with experimental evaluation, Information and Software Technology 52 (12) (2010) 1286–1297.

[40] K. Bulut, G. Jourdan, U. C. Türker, Minimizing characterizing sets, in: 16th International Conference on Formal Aspects of Component Software (FACS 2019), Vol. 12018 of Lecture Notes in Computer Science, Springer, 2019, pp. 72–86.

[41] D. Lee, M. Yannakakis, Testing finite-state machines: State identification and verification, IEEE Transactions on Computers 43 (3) (1994) 306–320.

[42] M. R. Garey, D. S. Johnson, Computers and Intractability, W. H. Freeman and Company, New York, 1979.

[43] F. Brglez, ACM/SIGMOD benchmark dataset, Available online at `https://people.engr.ncsu.edu/brglez/CBL/benchmarks/Benchmarks-upto-1996.html`, accessed: 2014-02-13 (1996).

[44] D. Neider, R. Smetsers, F. W. Vaandrager, H. Kuppens, Benchmarks for automata learning and conformance testing, Vol. 11200 of Lecture Notes in Computer Science, Springer, 2018, pp. 390–416.

[45] C. Güniçen, U. C. Türker, H. Ural, H. Yenigün, Generating preset distinguishing sequences using SAT, in: Computer and Information Sciences II, Springer London, 2012, pp. 487–493.

[46] F. Wilcoxon, Individual Comparisons by Ranking Methods, Springer New York, New York, NY, 1992, pp. 196–202.

[47] P. Teetor, R Cookbook, 1st Edition, O'Reilly, 2011.

[48] J. Cohen, Statistical power analysis for the behavioral sciences, Routledge, 1988.

[49] M. Soucha, Finite-state machine state identification sequences, Available online at `https://cyber.felk.cvut.cz/theses/papers/555.pdf`, accessed: 2020-12-06 (1996).

[50] U. Türker, H. Yenigün, Hardness and inapproximability of minimizing adaptive distinguishing sequences, Formal Methods in System Design 44 (3) (2014) 264–294.

[51] U. Türker, T. Ünlüyurt, H. Yenigün, Lookahead-based approaches for minimizing adaptive distinguishing sequences, in: Testing Software and Systems - 26th IFIP WG 6.1 International Conference, ICTSS 2014, Madrid, Spain, September 23-25, 2014. Proceedings, 2014, pp. 32–47.

[52] K. Naik, Efficient computation of unique input/output sequences in finite-state machines, IEEE/ACM Transactions on Networking 5 (4) (1997) 585–599.

[53] T. Y. Chen, M. F. Lau, Dividing strategies for the optimization of a test suite, Information Processing Letters 60 (3) (1996) 135 – 141.

[54] G. Rothermel, M. J. Harrold, Analyzing regression test selection techniques, IEEE Transactions on Software Engineering 22 (8) (1996) 529–551.

[55] G. Rothermel, M. J. Harrold, J. von Ronne, C. Hong, Empirical studies of test-suite reduction, Software Testing, Verification and Reliability 12 (4) (2002) 219–249.

[56] Z. Li, M. Harman, R. M. Hierons, Search algorithms for regression test case prioritization, IEEE Transactions on Software Engineering 33 (4) (2007) 225–237.

[57] S. Yoo, M. Harman, Regression testing minimization, selection and prioritization: a survey, Software Testing, Verification and Reliability 22 (2) (2012) 67–120.

[58] A. Petrenko, N. Yevtushenko, A. Lebedev, A. Das, Nondeterministic state machines in protocol conformance testing, in: Proceedings of Protocol Test Systems, VI (C-19), Elsevier Science (North-Holland), Pau, France, 1994, pp. 363–378.

[59] K. El-Fakih, R. Dorofeeva, N. Yevtushenko, G. V. Bochmann, FSM-based testing from user defined faults adapted to incremental and mutation testing, Programming and Computer Software 4/38 (2012) 1608–3261.