Incomplete Distinguishing Sequences for Finite State Machines

Robert M. Hierons[1] and Uraz Cengiz Türker[2]

[1]*Department of Computer Science, Brunel University London, Uxbridge, Middlesex, UK*
[2]*Sabanci University, Orhanli, Tuzla, 34956 Istanbul-Turkey*
Email: urazc@sabanciuniv.edu

Given a Finite State Machine (FSM) M, a *Distinguishing Sequence* (DS) is a test that identifies the state of M. While there are two types of DSs, preset DSs (PDSs) and adaptive DSs (ADSs), not all FSMs possess a DS. In this paper, we examine the problem of finding incomplete PDSs and ADSs, exploring associated optimisation problems: finding a largest set of states that has a DS and finding a smallest set of DSs that, between them, distinguish all of the states. We also propose a greedy algorithm to produce a small set of incomplete ADSs and use experiments to compare this with two previously published algorithms for generating state identifiers. We show that the optimisation problems related to incomplete ADSs and PDSs are PSPACE-complete **as are corresponding approximation problems. In the experiments we found that incomplete ADSs produced by the proposed greedy algorithm led to relatively compact state identifiers.**

Keywords: Model Based Testing, Finite State Machines, Testing, Checking Experiments, Adaptive Distinguishing Sequences

Received 00 January 2009; revised 00 Month 2009

1. INTRODUCTION

Software testing is an important part of the software development process but is often manual, expensive and error prone. This has led to much interest in test automation, including work on model based testing (MBT) where testing is automated on the basis of a model. Most MBT techniques and tools use behavioural models and typically operate on either a finite state machine (FSM) or a labelled transition system (LTS) that defines the semantics of the model used. There has thus been significant interest in automating testing based on an FSM or LTS model in areas such as sequential circuits [1], lexical analysis [2], software design [3], communication protocols [4], object-oriented systems [5], and web services [6] (see, in addition, [7, 8, 9, 10]). Such techniques have been shown to be effective when used in significant industrial projects [11]. We focus on testing from an FSM that is deterministic, minimal and completely-specified.

Most approaches to generating tests from an FSM can be seen as processes that test the transitions of the *system under test (SUT)*, and a crucial part of testing a transition τ is identifying the starting and ending states of τ. This problem is known as the *State Identification Problem.* Many techniques for constructing tests use distinguishing sequences (DSs) to resolve the state identification problem for two reasons: There are polynomial time algorithms that generate tests when there is a known DS and the length of the test is relatively short when designed with a DS [12, 13, 14, 15, 16].

In this paper we use the term *complete DS* to denote the usual notion of a DS; one that distinguishes all of the states of the FSM from which tests are being derived. Although complete DSs have a number of advantages over other approaches used to distinguish states, not all FSMs possess a complete DS. In this paper we consider the case where the FSM M does not have a complete DS and thus we would like to form a collection of *incomplete DSs* that, between them, distinguish all of the states of M.

Interest in the state identification problem has largely been motivated by checking experiment generation and fault localisation. A checking experiment is a test that is guaranteed to distinguish between the specification and the SUT if the SUT is faulty and satisfies certain well-defined conditions (typically an upper bound on the number of states of the SUT). Most techniques that generate checking experiments use strategies for solving the state identification problem: they typically either use a DS [17], a characterisation set (W-set) [3, 18], or harmonised state identifiers (HSIs) [19]. However, approaches that do not use DSs typically lead to significantly longer tests [20]. In fault localisation the problem is to determine the fault in the SUT that caused an observed failure and, again, there are efficient solutions when there is a known DS [21]. The motivation for the work reported in this paper comes primarily from the desire to obtain some of the benefits of complete DSs when a specification FSM does not have a complete DS.

A distinguishing sequence can be preset or adaptive: if the input sequence is fixed then it is a *Preset Distinguishing Sequence* (PDS) and otherwise, when the next input to be applied is determined based on the response to the previous input, it is an *Adaptive Distinguishing Sequence*[3] (ADS). Throughout the paper we refer to PDS or ADS when we write DS.

In Section 5 we consider problems associated with PDSs, where an input sequence is a PDS for a set $\bar{S}$ of states if it distinguishes the states in $\bar{S}$. The work in this section is motivated by the fact that sometimes we require tests that are not adaptive. We study the following question.

Definition 1.1 (MaxSubSetPDS problem). *Given deterministic, minimal and completely-specified FSM M with state set S and $\bar{S} \subseteq S$, find a subset $\bar{S}'$ of $\bar{S}$ that has a* PDS *such that $|\bar{S}'|$ is maximised.*

One way of expressing the problem of looking for a set of PDSs to distinguish all of the states of an FSM M with state set S is to look for a set P_S of subsets of S such that the following hold.

- For every pair of states $s, s' \in S$ with $s \neq s'$ there is some $\bar{S} \in P_S$ such that $s, s' \in \bar{S}$; and
- for every $\bar{S} \in P_S$ there is some PDS that distinguishes all of the states of $\bar{S}$.

This leads to the following definition of the MinSetPDS problem.

Definition 1.2 (MinSetPDS problem). *Given deterministic, minimal and completely-specified FSM M with state set S, find a set P_S of subsets of S such that each set in P_S has a* PDS*, for all $s, s' \in S$ with $s \neq s'$ we have that there exists $\bar{S} \in P_S$ such that $s, s' \in \bar{S}$, and this choice of P_S minimises $|P_S|$.*

The MaxSubSetPDS problem is motivated by the situation in which we have an expected current state s but we believe that the state is from some set $\bar{S}$: we want to distinguish the expected state from as many states in $\bar{S}$ as possible. The MinSetPDS problem is motivated by the desire to use as few incomplete DSs as possible to identify the state of M. When we require more than one incomplete DS to identify a state we have to run multiple tests, separated by resets. It has been observed that resets can be hard to realise and expensive to apply since they may require a complex system to be reinitialised or may require manual involvement [23, 24]. This has led to work that aims to minimise the number of input sequences (and so resets) used [25, 26, 27]. A tester might thus be particularly interest in the MinSetPDS problem.

We show that the MaxSubSetPDS and MinSetPDS problems are PSPACE-complete. Moreover, we show that the MaxSubSetPDS problem is inapproximable. In Section 6 we adapt the problems introduced so far to ADSs.

Definition 1.3 (MaxSubSetADS problem). *Given deterministic, minimal and completely-specified FSM M with state set S and $\bar{S} \subseteq S$, find a subset $\bar{S}'$ of $\bar{S}$ that has an* ADS *such that $|\bar{S}'|$ is maximised.*

Definition 1.4 (MinSetADS problem). *Given deterministic, minimal and completely-specified FSM M with state set S, find a set P_S of subsets of S such that each set in P_S has an* ADS*, for all $s, s' \in S$ with $s \neq s'$ we have that there exists $\bar{S} \in P_S$ such that $s, s' \in \bar{S}$, and this choice of P_S minimises $|P_S|$.*

We show that the MaxSubSetADS and MinSetADS problems are PSPACE-complete. We also show that the MaxSubSetADS problem is inapproximable. This contrasts with looking for a complete ADS, a problem that can be solved in polynomial time.

Having determined the complexity of these problems, we propose a greedy algorithm for the MinSetADS problem and report on the results of experiments. The experiments used a set of FSMs and compared the state identification sequences computed by three approaches: a method for generating a characterisation set (W-generation method) [28]; a method for generating harmonised state identifiers (HSI-generation method) [19]; and state identification sequences derived from incomplete ADSs generated using the greedy algorithm (ADS-method). Note that the state identification sequences derived from incomplete ADSs define a set of harmonised state identifiers and so the comparison with the HSI-generation method [19] is a comparison between two approaches for generating an HSI-set. The experimental subjects included randomly generated FSMs and FSMs drawn from a benchmark and suggest that the ADS-method produces compact state identification sequences.

The paper is organised as follows. In the next section we review related work and in Section 3 we define the terminology and notation used throughout the paper. In Section 4 we introduce a motivating example in which we demonstrate what we can gain by using incomplete ADSs for state identification. In Section 5 we present results related to PDSs and subsequently, in Section 6, we present results related to ADSs. In Section 7 we report on the results of experiments. In Section 8, we conclude the paper and discuss some possible lines of future work.

2. RELATED WORK

This section reviews previous work on state identification sequences, starting with DSs. There are many

[3]ADSs are also called *Distinguishing Sets* [12, 22].

computational complexity results regarding DSs. It was show that checking the existence of a PDS is a PSPACE-complete problem [16]. Although earlier bounds for ADSs are exponential in the number of states [29], Sokolovskii proved that if an FSM M with n states has an ADS then it has an ADS with height no greater than $\pi^2 n^2/12$ [30]. Moreover, Kogan claimed that for a given n state FSM, the length of an ADS is bounded above by $n(n-1)/2$ [31] and later Rystsov proved this claim [32]. More than twenty years later, Lee and Yannakakis proposed an algorithm that constructs an ADS with upper bound of $n(n-1)/2$ in the worst case [16]. Kushik et al. present an algorithm for constructing ADSs for nondeterministic observable FSMs [33]. Since the class of deterministic FSMs is a subclass of nondeterministic observable FSMs, the algorithm can also be used to construct ADSs for a given FSM M. Recently Türker et al. presented a lookahead based algorithm to construct compact ADSs [34].

Unfortunately, not all FSMs possess a DS. For such cases, Kohavi et al. [35] suggested that the states of the FSM should be partitioned in such a way that a DS exists for each set of states. Lee and Yannakakis showed that checking the existence of an ADS for a set of states is PSPACE-complete [16]. However, as far as we know there has been no previous work on other problems related to generating a set of DSs that distinguishes every state of the FSM. In addition, we are not aware of methods that generate a set of DSs that distinguishes every state of the FSM.

For FSMs that do not have a DS, the state identification problem can sometimes be solved by using unique input output sequences (UIOs) [36]. A UIO for a state s of FSM M distinguishes s from all other states of M but need not distinguish other states. When an FSM does not have a UIO, a partial UIO (pUIO) can be constructed [37], where a pUIO for state s distinguishes state s from a subset of states. Some FSMs do not have either a DS or a UIO for each state. However, every minimal FSM with n states and m inputs does have a characterisation set (W-set) which can be computed in time of $O(n^2m)$[3, 18, 29].

It has long been known that FSM specifications are sometimes partial (partial FSMs): that some state-input combinations do not have corresponding transitions [38, 39, 40, 41]. Recently, it was shown that for partial FSMs, checking the existence of an ADS can be done in polynomial time and checking the existence of a PDS is PSPACE-complete [42].

As in the case of complete FSMs, a partial FSM need not have an ADS or a PDS. In addition, a partial FSM need not have a W-set. In this case, the state identification problem can be resolved by using harmonised state identifiers [19]. A set of harmonised state identifiers is actually a partial characterisation set where for any two states s and s', the corresponding sets have sequences w and w' that have a common prefix that distinguishes s and s'. For a completely specified FSM with n states and m inputs, harmonised state identifiers can be computed in time of $O(n^4m)$ [19]. However, for partially specified FSMs, harmonised state identifiers can be computed in time of $O(n^5m)$.

DSs, UIOs, W-sets and HSIs are interesting in their own right in offering a solution to the state identification problem. However, it has been shown that these sequences are building blocks for solutions to another important problem: *fault detection.*

In the fault detection problem we are given a specification FSM M, and an unknown (black-box) implementation FSM N, and we want to check whether N is a correct implementation of M. The fault detection problem was first posed in Moore's seminal work [43]. In fault detection, we usually want to construct a test that can detect faulty transitions in the implementation machine. To obtain good fault coverage, we want to identify the ending state of a transition during a test. Hennie showed that a PDS can be used to construct a fault detection sequence [44]. Later, it was shown that instead of a PDS an ADS can be used in the same method [45]. Later, methods for constructing a fault detection sequence using UIOs and W-sets were proposed [3, 18, 46].

As the length of the fault detection sequence determines the duration and hence the cost of testing, in the literature there exists a line of work that aims to reduce the length (number of inputs) of fault detection sequences. These approaches identify a set of sequences that need to be included and aim to generate a short fault detection sequence that contains these sequences. Important methods include the DS [44, 47], W [3, 18], Wp [24], UIO [36], UIOv [48], HSI [19], SPY [49], H [50] and P [27] test derivation methods. As these methods rely on heuristics, they produce different results for different types of FSMs and it is difficult to compare them analytically. Consequently, experimental evaluation has been conducted to identify trade-offs among these methods [51, 52].

3. PRELIMINARIES

An FSM has a finite set of states and transitions between the states, with transitions being labelled with input/output pairs.

Definition 3.1. *An FSM (or Mealy machine) M is defined by a tuple $(S, X, Y, \delta, \lambda, s_0)$ where $S = \{s_1, s_2, \ldots s_n\}$ is the finite set of states, $X = \{a, b, \ldots, p\}$ and $Y = \{1, 2, \ldots, q\}$ are the finite sets of inputs and outputs, $\delta : S \times X \to S$ is the state transition function, $\lambda : S \times X \to Y$ is the output function, and $s_0 \in S$ is the initial state.*

If FSM M is in state $s \in S$ and input $x \in X$ is applied then M moves to the state $s' = \delta(s, x)$ and produces output $y = \lambda(s, x)$. Such a *transition* will be denoted $\tau = (s, x/y, s')$ and we say that x/y is the *label* of τ ($label(\tau)$), s is the *start state* of τ ($start(\tau)$), and s' is the *end state* of τ ($end(\tau)$). An FSM M can be represented by a directed graph G, where the states of M are represented by corresponding vertices of G and the transitions of M are represented by edges of G. An edge is labelled by the input/output pair of the corresponding transition and so if $\delta(s, x) = s'$ and $\lambda(s, x) = y$, then the edge corresponding to this transition has label x/y and ends at the vertex corresponding to s'. Figure 1 represents an FSM in which $S = \{s_1, s_2, s_3\}$, $X = \{a, b\}$, $Y = \{0, 1\}$, and the initial state is s_1 (highlighted with a dashed circle).

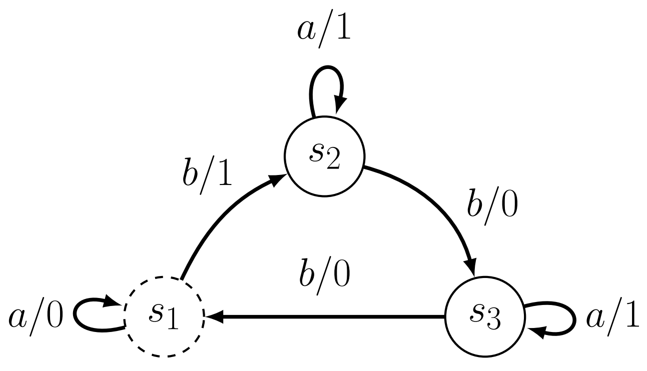

FIGURE 1: An example FSM M_0 with initial state s_1.

We use juxtaposition to denote concatenation: if x_1, x_2, and x_3 are inputs, $x_1x_2x_3$ is an input sequence. Given a set X we let X^* denote the set of sequences of elements of X and let X^k denote the sequences in X^* that have length k. The symbol ε is used to denote the empty sequence. The transition function and the output function can be extended to sequences of inputs. In an abuse of notation, we use δ and λ for the extended functions. These extensions are defined as follows in which $x \in X$ and $w \in X^\star$: $\delta(s, \varepsilon) = s$ and $\delta(xw) = \delta(\delta(s, x), w)$; $\lambda(s, \varepsilon) = \varepsilon$ and $\lambda(s, xw) = \lambda(s, x)\lambda(\delta(s, x), w)$. Moreover, we use δ and λ when we consider transitions from a set of states: given $\bar{S} \subseteq S$ and $w \in X^\star$, we define $\delta(\bar{S}, w) = \cup_{s\in\bar{S}}\{\delta(s, w)\}$ and $\lambda(\bar{S}, w) = \cup_{s\in\bar{S}}\{\lambda(s, w)\}$. Two states $s, s' \in S$ are said to be *equivalent* if for all input sequences $\alpha \in X^\star, \lambda(s, \alpha) = \lambda(s', \alpha)$. If there exists an input sequence $\alpha \in X^\star$ such that $\lambda(s, \alpha) \neq \lambda(s', \alpha)$, then s and s' are said to be *distinguishable*. An FSM M is *minimal* if the states of M are pairwise distinguishable. M is *completely-specified* if both δ and λ are total functions (they are defined on all state/input pairs). Since δ and λ are functions (rather than relations), there can be at most one transition defined for each state/input pair. Such machines are said to be *deterministic*.

An input/output sequence consists of a sequence of input/output pairs of the form $x_1/y_1x_2/y_2 \dots x_m/y_m$. We will also write $x_1x_2 \dots x_m/y_1y_2 \dots y_m$ to denote the same input/output sequence, where $x_1x_2 \dots x_m$ is called the *input portion* and $y_1y_2 \dots y_m$ is called the *output portion* of the input/output sequence. A *path* in M is a sequence of transitions $\bar{\tau} = \tau_1\tau_2 \dots \tau_m$ such that $start(\tau_i) = end(\tau_{i-1})$, for all $1 < i \leq m$. The *label* of a path is an input/output sequence which is the concatenation of the labels (input/output pairs) of the transitions in that path. For $\bar{\tau} = \tau_1\tau_2 \dots \tau_m$, we define $label(\bar{\tau}) = label(\tau_1)label(\tau_2) \dots label(\tau_m)$. An example of a path in M_0 of Figure 1 is $\bar{\tau} = (s_1, b/1, s_2)(s_2, a/1, s_2)$, and we have $label(\bar{\tau}) = b/1a/1$.

In this work, we consider only deterministic, completely-specified, minimal FSMs. An FSM can be minimised in polynomial time [53]. Further, an FSM that is not completely-specified can often be completed by adding either an error state or transitions with null output[4]. For example, in Harel statecharts if input x is received in state s and there is no specified transition then x leads to no change in state and no output (a null transition). The requirement that the specification and implementation are deterministic is one made by many FSM based testing methods and one that holds in important application domains such as protocols and (many) embedded systems[5]. Given two states $s \neq s'$, an input sequence w is said to be a *separating sequence* for s and s', if $\lambda(s, w) \neq \lambda(s', w)$. A *characterisation set* is a set of input sequences (W) such that for any pair of states $s, s' \in S$ there exists an input sequence $w \in W$ such that $\lambda(s, w) \neq \lambda(s', w)$. The set $W_s \subseteq W$ is said to be a *state identifier* for s if for all $s' \in S \setminus \{s\}$ there exists an input sequence $w \in W_s$ such that $\lambda(s, w) \neq \lambda(s', w)$. Note that a separating sequence that distinguishes states s and s' is also a PDS for the state set $\{s, s'\}$.

Given deterministic, minimal, completely-specified FSM M with state set $S = \{s_1, s_2, \dots, s_n\}$, the sets $H_1, H_2, \dots, H_n$ of input sequences are *harmonised state identifiers* if and only if for all $1 \leq i, j \leq n$ with $i \neq j$ there exist input sequences $w_i \in H_i$ and $w_j \in H_j$ such that a common prefix w of w_i and w_j distinguishes s_i and s_j ($\lambda(s_i, w) \neq \lambda(s_j, w)$).

We now define Preset and Adaptive Distinguishing Sequences.

Definition 3.2. *Given deterministic, minimal and completely-specified FSM $M = (S, X, Y, \delta, \lambda)$ and $\bar{S} \subseteq S$, input sequence w is a* Preset Distinguishing Sequence (PDS) *for $\bar{S}$ if for all $s, s' \in \bar{S}$ with $s \neq s'$ we have that $\lambda(s, w) \neq \lambda(s', w)$.*

If w is a PDS for state set S then we say that it is a *complete PDS*. Otherwise, w is an *incomplete PDS* for $\bar{S}$.

Adaptive distinguishing sequences are finite trees

[4]As has been previously noted, it is not always possible to complete an FSM since, for example, unspecified input may correspond to input that should not occur [54].

[5]Most FSM based test methods apply to deterministic FSMs but there are situations in which a specification will be non-deterministic and here we require different test generation methods [55, 24, 56, 19, 57, 58, 59].

rather than sequences. An ADS $\mathcal{A}$ for state set $\bar{S}$ has an initial node, which corresponds to the situation before the ADS has been applied. All edges from the initial node v_1 are labelled with the same input x_1 and when $\mathcal{A}$ is used the input x_1 is applied first. If the response to x_1 is output y_1 then $\mathcal{A}$ moves to the node v_2 that is reached from v_1 by an edge with label x_1/y_1. If v_2 is a leaf then the application of $\mathcal{A}$ is complete but otherwise the input x_2 that is on edges from v_2 is applied to the SUT next, the output y_2 produced is recorded, and the edge of $\mathcal{A}$ from v_2 with label x_2/y_2 is followed. This process continues until a leaf is reached. An ADS for state set $\bar{S}$ of FSM M leads to different input/output sequences from distinct states of $\bar{S}$ and so it distinguishes these states. More formally, we define ADSs as follows.

Definition 3.3. *An* Adaptive Distinguishing Sequence (ADS) *for a state set $\bar{S}$ with m states is a rooted tree $\mathcal{A}$ with exactly m leaves; the edges are labeled with an input/output pair and the leaves are labeled with a single state such that: 1) input labels of edges leaving a common node are the same and output labels of edges leaving a common node are different. 2) for every leaf of $\mathcal{A}$, if $\bar{x}$, $\bar{y}$ are the input output sequences respectively formed by the edge labels on the path from the root node to the leaf and if the leaf is labeled by a single state s, then $\lambda(s, \bar{x}) = \bar{y}$.*

If $\mathcal{A}$ is an ADS for state set S then we say that it is a *complete ADS*. Otherwise, $\mathcal{A}$ is an *incomplete ADS* for $\bar{S}$.

An ADS for $\bar{S}$ defines an experiment where the next input to be applied depends on the previously observed input/output sequence (and so the node reached). If we apply $\mathcal{A}$ in a state $s \in \bar{S}$ then the resultant input/output sequence is that which labels the path of $\mathcal{A}$ from the root of $\mathcal{A}$ to a leaf and is also the label of a path of M that has starting state s. By the definition of an ADS the input/output sequences for two distinct states from $\bar{S}$ must differ and so $\mathcal{A}$ distinguishes the states from $\bar{S}$. Throughout the paper we refer to the depth of ADS tree $\mathcal{A}$ when we write the length of $\mathcal{A}$.

Note that when we set $\bar{S} = S$, Definitions 3.2 and 3.3 correspond to the classical notions of Preset and Adaptive Distinguishing sequences.

We present an example FSM, which will be used throughout the paper, in Figure 2. We also present a manually computed incomplete ADS for states s_1, s_2 and s_4 in Figure 3.

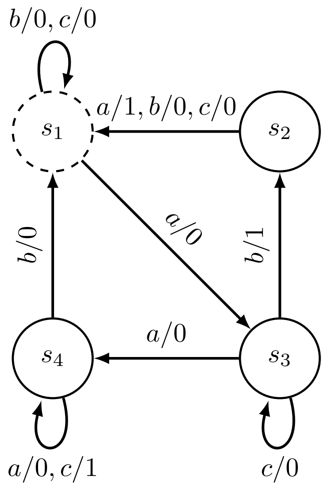

FIGURE 2: An example FSM M_1 with initial state s_1.

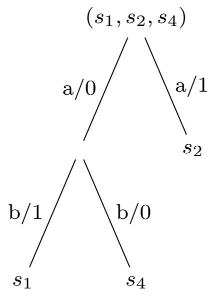

FIGURE 3: An incomplete ADS for machine M_1 presented in Figure 2 where $\bar{S} = \{s_1, s_2, s_4\}$.

The adaptive experiment starts with input a: if the underlying FSM produces 1 then the adaptive experiment ends and the tester deduces that the FSM was in state s_2, otherwise the tester will apply an input b and if it observes output 0 then it decides that the FSM was in state s_4 and otherwise it deduces that the FSM was in state s_1.

Let us suppose that we have a set $A = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k\}$ of incomplete ADSs such that every pair of distinct states of M is distinguished by some ADS from A; such a set will be said to be *fully distinguishing*. Every pair of states $s, s' \in \bar{S}$ with $s \neq s'$ is distinguished by at least one ADS in $A = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k\}$. As noted earlier, we can represent this as there being a set $P_S = \{\bar{S}_1, \bar{S}_2, \ldots, \bar{S}_k\}$ of subsets of $\bar{S}$ such that for all $s, s' \in \bar{S}$ there exists $1 \leq j \leq k$ such that $s, s' \in \bar{S}_j$ and there is an ADS $\mathcal{A}_j$ for $\bar{S}_j$. In this situation, in order to distinguish a state $s \in \bar{S}$ from other states in $\bar{S}$ we use every $\mathcal{A}_j$ such that $s \in \bar{S}_j$.

Given state $s_i \in \bar{S}$ we will let $A(s_i)$ denote the set of ADSs that are to be used to distinguish s_i from other states in $\bar{S}$: the set of $\mathcal{A}_j$ such that $s_i \in \bar{S}_j$. Note that $\bar{S}_j$ need not be the largest subset of $\bar{S}$ that contains states that are distinguished from s_i by $\mathcal{A}_j$ since some pairs of states may be distinguished by more than one ADS to be used but we only require this pair of states to be in one $\bar{S}_j$. We will also let $H(s_i, \mathcal{A}_j) \in X^*$ denote the input portion of the input/output sequence produced when $\mathcal{A}_j$ is applied in state s_i. Given state s_i, we will let $H_i(A)$ be the set of maximal sequences in $\{H(s_i, \mathcal{A}_j) | \mathcal{A}_j \in A(s_i)\}$ (by maximal we mean that if $H(s_i, \mathcal{A}_j) = \bar{x}$ and $H(s_i, \mathcal{A}_{j'}) = \bar{x}'$ for $\mathcal{A}_j, \mathcal{A}_{j'} \in A(s_i)$ and $\bar{x}'$ is a proper prefix of $\bar{x}$ then we do not include $\bar{x}'$). Then $H_i(A)$ is the set of input sequences applied when using ADSs from $A(s_i)$ in state s_i. We obtain the following result that shows how ADSs and harmonised state identifiers relate.

Proposition 3.1. *Given deterministic, minimal and completely-specified FSM M and fully distinguishing set $A = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k\}$ for M, the $H_i(A)$ are harmonised state identifiers for M.*

Proof. It is sufficient to prove that if s_i, s_j are distinct states of M then there are input sequences $w_i \in H_i(A)$ and $w_j \in H_j(A)$ such that there is a common prefix w of w_i and w_j that distinguishes s_i and s_j. First observe that since A is fully distinguishing there is some $\mathcal{A}_l \in A$ that distinguishes s_i and s_j. But by definition this means that the application of $\mathcal{A}_l$ from s_i and s_j leads to different input/output sequences. However, the input sequence can only differ once a different output has been observed. Thus, $\mathcal{A}_l$ has a node v such that the following hold:

1. The path from the root of $\mathcal{A}_l$ to v has a label α/β that labels paths from both s_i and s_j; and
2. There are edges with labels x/y_i and x/y_j from v with $y_i \neq y_j$ such that $\alpha x/\beta y_i$ labels a path from s_i and $\alpha x/\beta y_j$ labels a path from s_j.

However, this means that $w = \alpha x$ is a prefix of input sequences in H_i and H_j and also that w distinguishes s_i and s_j. The result therefore follows. □

4. MOTIVATING EXAMPLE

We manually computed the fully distinguishing set $A = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4\}$ for FSM M_1 given in Figure 2. The incomplete ADSs are given in Figure 4. The resulting harmonised state identifiers are given as follows: $H_1(A) = \{ab, b\}, H_2(A) = \{a, b\}, H_3(A) = \{b\}$ and $H_4(A) = \{ab, b\}$.

For the FSM presented in Figure 2 the characterisation sets (according to the algorithm presented in [29]) are given as $W = \{a, b, c\}$ and so the state identifiers for M are $W_1 = \{a, b, c\}$, $W_2 = \{a, b, c\}$, $W_3 = \{a, b, c\}$, and $W_4 = \{a, b, c\}$. According to the algorithm given in [19], if we use the characterisation sets W for M_1 then the harmonised state identifiers for the FSM presented in Figure 2 are: $H_1 = \{a, b, c\}$, $H_2 = \{a\}$, $H_3 = \{a, b\}$, and $H_4 = \{a, b, c\}$.

From now on for a given FSM M, $\mathcal{SI}$ refers to the average number of sequences in the state identifiers/harmonised state identifiers and $\mathcal{LI}$ depicts

	ADS	HSI	W
$\mathcal{SI}$	1.75	2.33	3
$\mathcal{LI}$	1.7	1	1

TABLE 1: $\mathcal{SI}$ and $\mathcal{LS}$ results with respect to ADS, HSI and W-set for FSM M_1.

the average length of the state identifiers/harmonised state identifiers. Given $K \in \{H, W\}$, the following define $\mathcal{SI}$ and $\mathcal{LI}$.

$$\mathcal{SI} = \frac{\sum_{1 \le i \le n} |K_i|}{n} \tag{1}$$

$$\mathcal{LI} = \frac{\sum_{1 \le i \le n} \sum_{1 \le j \le |K_i|} |w_j|}{\sum_{1 \le i \le n} |K_i|} \tag{2}$$

where n is the number of states of FSM. For the FSM given in Figure 2 the $\mathcal{SI}$ and $\mathcal{LI}$ values are computed and the results are given in Table 1.

In this simple example, the results suggest that the use of incomplete ADSs can reduce the average number of state identifiers by 25% compared to the HSI-generation method and by 42% compared to the W-generation method. We also notice that the average length of distinguishing sequences is relatively high when incomplete ADSs are used. In Section 7 we show that this is not generally the case.

5. INCOMPLETE PRESET DISTINGUISHING SEQUENCES

We first introduce some basic terminology that we use throughout this section.

Definition 5.1. *A* Finite Automaton *(FA) is defined by a tuple $A = (Q, \Sigma, \delta, 0, F)$ where Q is the finite set of states, Σ is the finite alphabet, δ is the transition function of type $Q \times \Sigma \to Q$, 0 is the initial state and $F \subseteq Q$ is the set of accepting states.*

Since δ is a function we implicitly refer to deterministic finite automata; whenever we use the term finite automaton we will be referring to a deterministic finite automaton. A word is accepted by finite automaton A, if and only if it takes A from 0 to an accepting state (a state in F). The set of all words accepted by a finite automaton A defines the (regular) language denoted $L(A)$. We assume that an FA A considered is minimal in the sense that there is no FA A' with fewer states than A such that $L(A') = L(A)$.

We show that the MaxSubSetPDS problem is PSPACE-complete through relating it to the Finite Automata Intersection Problem, which was introduced by Dexter Kozen and is PSPACE-complete [60].

Definition 5.2 (Finite Automata Intersection Problem (FA-INT)). *Let $\mathbb{A} = \{A_1, A_2, \ldots, A_z\}$ be z finite automata with a common alphabet Σ. The FA-INT problem is to determine whether the A_i accept a*

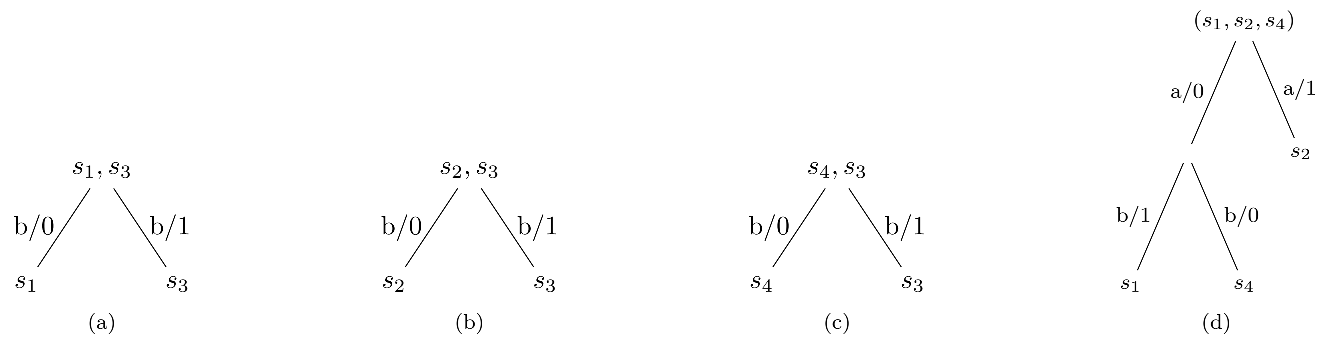

FIGURE 4: Incomplete ADSs for machine M_1 presented in Figure 2. Incomplete ADS $\mathcal{A}_1$ that distinguishes pair of states (s_1, s_3) (Figure 4a). Incomplete ADS $\mathcal{A}_2$ that distinguishes pair of states (s_2, s_3) (Figure 4b). Incomplete ADS $\mathcal{A}_3$ that distinguishes pair of states (s_4, s_3) (Figure 4c). Incomplete ADS $\mathcal{A}_4$ that distinguishes pair of states (s_1, s_2), (s_1, s_4), and (s_2, s_4) (Figure 4d).

common element of $\Sigma^\star$, i.e. whether there is a word w such that $w \in L(A_i)$ for all $1 \leq i \leq z$.

It is straightforward to see that the complexity of the FA-INT problem is not altered if we restrict attention to non-empty words since we can decide whether all of the A_i accept ε in polynomial time. In addition, since a FA can be minimised in polynomial time, the complexity of the FA-INT problem is not affected by only considering minimal FA.

Without loss of generality we assume that the finite automata in $\mathbb{A}$ have disjoint sets of states. Given an instance of the FA-INT problem, with a finite set $\mathbb{A} = \{A_1, A_2, \ldots, A_z\}$ of finite automata on a common finite alphabet Σ ($A_i = (Q_i, \Sigma, \delta_i, 0_i, F_i)$), we construct an FSM $\mathcal{M}_1(\mathbb{A}) = (S, X, Y, \delta, \lambda, s_0)$ as follows (this construction is similar to one in [16]).

We introduce a new state *Sink*. Then we take two copies $A_i^1 = (Q_i^1, \Sigma, \delta_i^1, 0_i^1, F_i^1), A_i^2 = (Q_i^2, \Sigma, \delta_i^2, 0_i^2, F_i^2)$ of each finite automaton A_i and call them *pair automata*. Given $q \in Q_i$ we let q^1 and q^2 denote the corresponding states in Q_i^1 and Q_i^2 respectively. We let $\bar{S} = \{0_1^1, 0_1^2, \ldots, 0_z^1, 0_z^2\}$, which is the set of initial states of the copies of the finite automata. The set of states of the FSM to be constructed is given by $S = Q_1^1 \cup Q_1^2 \cup Q_2^1 \cup Q_2^2 \cup \ldots \cup Q_z^2 \cup \{Sink\}$, where the initial state is selected as 0_1^1. The input alphabet of the FSM is given by $X = \Sigma \cup \{D\}$ for an additional input D whose use will be explained below. The output alphabet of the FSM is given by $Y = Q_1 \cup Q_2 \cup \ldots \cup Q_z \cup \{0, 1, 2\}$.

The state transitions of the finite automata in $\mathbb{A}$ are inherited: if $a \in \Sigma$ and $q_i^j \in Q_i^j$ for $1 \leq i \leq z$ and $1 \leq j \leq 2$ then $\delta(q_i^j, a) = r_i^j$ for the state r_i of A_i such that $\delta_i(q_i, a) = r_i$. Input D takes all states to *Sink*.

The output function λ of $\mathcal{M}_1(\mathbb{A})$ is defined as follows, in which $1 \leq i \leq z$.

$$\lambda(s, x) = \begin{cases} q_i, & \text{If } x \neq D \text{ and } s = q_i^j \text{ for some } \\ & q_i^j \in Q_i^1 \cup Q_i^2, \\ q_i, & \text{If } x = D \text{ and } s = q_i^j \text{ for some } \\ & q_i^j \in (Q_i^1 \cup Q_i^2) \setminus (F_i^1 \cup F_i^2), \\ 0, & \text{If } s = Sink, \\ 1, & \text{If } x = D \text{ and } s \in F_i^1, \\ 2, & \text{If } x = D \text{ and } s \in F_i^2. \end{cases}$$

There are states of this FSM that cannot be reached from the initial state. This does not affect the proof but we could choose to extend this FSM by, for example, for $1 \leq i \leq z$ and $j \in \{1, 2\}$ adding an input x_i^j that takes all states to state 0_i^j with output y for some fixed y. We illustrate the construction in Figure 5 in which we include two copies of the state *Sink* to aid readability.

The basic idea is that until D is received the transitions from a state in $Q_i^1 \cup Q_i^2$ simulate the state transitions of A_i but also tell us which states of A_i are being traversed and so the value of i (the A_i have disjoint state sets). If D is received in a state q_i^j from Q_i^j then the output tells us the value of j if and only if the state q_i^j is such that q_i is an accepting state of A_i. We now explore properties of $\mathcal{M}_1(\mathbb{A})$, proving results that will be brought together in Theorem 5.1.

LEMMA 5.1. *Let us suppose that set $\mathbb{A} = \{A_1, A_2, \ldots, A_z\}$ of finite automata have a common alphabet Σ. The FSM $\mathcal{M}_1(\mathbb{A}) = (S, X, Y, \delta, \lambda, s_0)$ has a* PDS *for $\bar{S} = \{0_1^1, 0_1^2, \ldots, 0_z^1, 0_z^2\}$ if and only if there is a non-empty word $\omega \in \Sigma^\star$ that is accepted by all of the finite automata (in which case ωD is such a* PDS*).*

Proof. First, let us suppose that $\omega \neq \varepsilon$ is in the intersections of the languages of the A_i and consider $w = \omega D$. By construction, ω distinguishes any two 0_i^α and 0_j^β with $i \neq j$ since $\omega \in \Sigma^\star$ is non-empty, the output in response to an element of Σ identifies the state of the corresponding A_i, and the state sets of the A_i are pairwise disjoint. Further, if we consider states 0_i^1 and

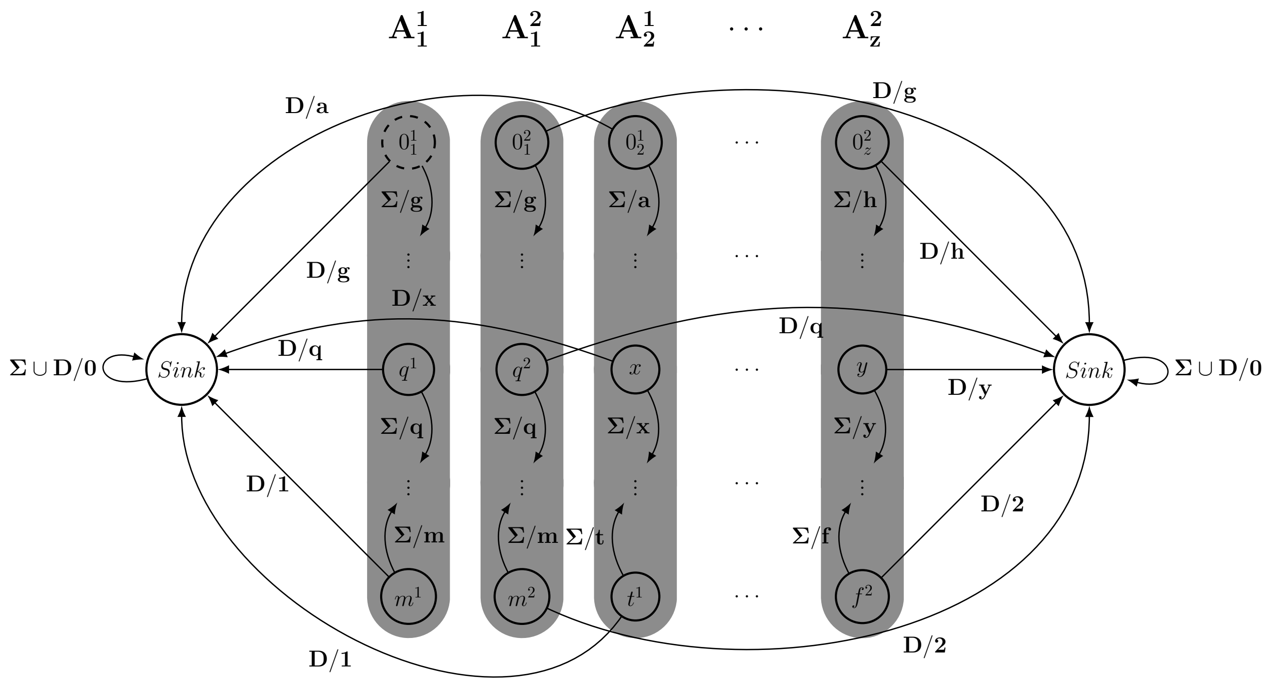

FIGURE 5: An FSM $\mathcal{M}_1(\mathbb{A})$ constructed from an FA-INT problem instance with $\bar{S} = \{0_1^1, 0_1^2, \ldots, 0_z^1, 0_z^2\}$, with initial state 0_1^1.

0_i^2 we find that ω takes them to accepting states from F_i and then D leads to different outputs (1 and 2). Thus, if there is some non-empty $\omega \in \Sigma^\star$ in the intersections of the languages of the A_i then $\mathcal{M}_1(\mathbb{A})$ has a PDS ωD for $\bar{S}$.

We now prove that if $\mathcal{M}_1(\mathbb{A})$ has a PDS for $\bar{S}$ then there is some non-empty $\omega \in \Sigma^\star$ in the intersection of the languages of the A_i. We can observe that in order to distinguish states 0_i^1 and 0_i^2 it is necessary to apply input D but also that after D has been applied the state must be *Sink* and further input cannot distinguish the states. Thus there is a PDS for $\bar{S}$ if and only if there is a PDS for $\bar{S}$ that has the form $w = \omega D$ where $\omega \in \Sigma^\star$ and we now consider such a PDS.

Now let us suppose that $\delta(0_i^1, \omega) \notin F_i^1$ for some $1 \leq i \leq z$. Then $\delta(\delta(0_i^1, \omega), D) = Sink$ and similarly $\delta(\delta(0_i^2, \omega), D) = Sink$ and it is clear that $\lambda(0_i^1, \omega D) = \lambda(0_i^2, \omega D)$. This contradicts ωD being a PDS for $\bar{S}$. Therefore w must be in the form $w = \omega D$ such that ω is non-empty and brings all the initial states to accepting states. Thus, if $\mathcal{M}_1(\mathbb{A})$ has a PDS for $\bar{S}$ then there is some non-empty $\omega \in \Sigma^\star$ in the intersections of the languages of the A_i. □

We now consider how a non-deterministic Turing Machine can decide whether there is a PDS for a given state set $\bar{S}$ of FSM M. In this process it guesses inputs one at a time and maintains a current set π of pairs of states such that (s, s') is in π if and only if $s \in \bar{S}$ and the sequences of inputs received takes M from s to s'. It also maintains an equivalence relation r between states from $\bar{S}$: two states s, s'' are related under r if they have not been distinguished by the input sequence w that has been chosen ($\lambda(s, w) = \lambda(s'', w)$). It is straightforward to see that these two pieces of information can be updated when a new input is received; we do not need to know the previous inputs received. Further, the input sequence received defines a PDS for $\bar{S}$ if and only if no two different states from $\bar{S}$ are related under r.

Lemma 5.2. *The problem of deciding whether a set $\bar{S}$ of states of deterministic, minimal and completely-specified FSM M has a* PDS *is in* PSPACE.

Proof. We will show that a non-deterministic Turing Machine can solve this using polynomial space. Such a machine will guess inputs one at a time. It will maintain the set π of pairs of states and equivalence relation r as described above and this uses polynomial space. After guessing a new input x and updating π and r the machine checks whether the input sequence received defines a PDS for $\bar{S}$: this is the case if and only if r relates no two different states of $\bar{S}$. Thus, if M has a PDS for $\bar{S}$ then this non-deterministic Turing Machine will find such a PDS using polynomial space.

We now have to consider the case where M does not have a PDS for $\bar{S}$: we require that the non-deterministic Turing Machine terminates. In order to ensure this we use the result that if M has n states and $\bar{S}$ has m states then M has a PDS for $\bar{S}$ if and only if it has such a PDS with length at most $B = (m-1)n^m$ [29][6]. The non-deterministic Turing Machine therefore includes a counter that counts how many inputs have been received: the machine terminates with failure if the counter exceeds the upper bound. We require additional $O(\log_2 B) = O(\log_2(m-1) + m\log_2(n)) = O(m\log_2(n))$ space for the counter and so the space required is still polynomial.

We have defined a non-deterministic Turing Machine that requires only polynomial space in order to solve the problem and so the problem is in non-deterministic PSPACE. We can now use Savitch's Theorem [61], which tells us that a problem is in PSPACE if and only if it is in non-deterministic PSPACE, and the result follows. □

Condon et al. introduced the maximisation version of the FA-INT problem [62].

Definition 5.3 (Maximisation of FA-INT Problem (MAX FA-INT)). *Let us suppose that $\mathbb{A} = \{A_1, A_2, \ldots, A_m\}$ is a set of finite automata with input alphabet Σ. The MAX FA-INT problem is: What is the largest k such that there are k finite automata from $\mathbb{A}$ that accept a common word $w \in \Sigma^\star$.*

They also proved that it is PSPACE-hard to approximate the MAX FA-INT problem. We claim that based on the approach used to prove Lemma 5.1, we can give a similar result for the MaxSubSetPDS problem but before this we explore the relationship between the optimum solutions of the MaxSubSetPDS and MAX FA-INT problems.

Below, given a property P (such as distinguishing k states of an FSM) a word w is said to be a minimal word satisfying P if w satisfies P and no proper prefix of w satisfies P. The following is clear from the proof of Lemma 5.1.

Lemma 5.3. *Given set $\mathbb{A}$ of finite automata, let $OPT_{\mathbb{A}}$ be the set of minimal words that are accepted by the maximum number of finite automata from $\mathbb{A}$. Further, given $\mathcal{M}_1(\mathbb{A})$ let $OPT_{\mathcal{M}_1(\mathbb{A})}$ be the set of minimal words that maximise the size of the subset of $\bar{S}$ whose states are pairwise distinguished. Then $w \in OPT_{\mathbb{A}}$ if and only if $wD \in OPT_{\mathcal{M}_1(\mathbb{A})}$.*

We can now show that the MaxSubSetPDS problem, of finding a PDS that distinguishes the most states from some set $\bar{S}$, is PSPACE-complete and inapproximable. This is important since it shows that the problem of finding a good approximation to the optimisation problem is also PSPACE-complete.

The following concerns approximating the MaxSubSetPDS problem. Here the approximation is with respect to the size of the set of states distinguished by the PDS returned by an algorithm (when compared to an optimal solution).

Theorem 5.1. *The MaxSubSetPDS problem is PSPACE-complete and there exists a constant $\varepsilon > 0$ such approximating the MaxSubSetPDS problem within ratio n^ε is PSPACE-hard.*

Proof. The problem being PSPACE-hard follows from Lemma 5.3 and the MAX FA-INT problems being PSPACE-hard. To see that this problem is in PSPACE, first observe that it is sufficient to prove that the following problem is in PSPACE: for $1 \leq k \leq n$ decide whether there is a PDS that distinguishes k states of the FSM M. We can show that this is in PSPACE in a similar manner to Lemma 5.2, the only differences being that in a first step the non-deterministic Turing Machine guesses the set $\bar{S}'$ of k states.

To prove that the problem of approximating the MaxSubSetPDS is PSPACE-hard, let us assume that we have an algorithm $\mathcal{P}$ that belongs to a complexity class $C < PSPACE$ and returns an n^ε approximation for the MaxSubSetPDS Problem. In such a case, given an instance $\mathbb{A}$ of the MAX FA-INT problem, we can construct FSM $\mathcal{M}_1(\mathbb{A})$ and using $\mathcal{P}$ we can obtain a solution $w = \omega D$. But then Lemma 5.3 implies that ω defines an approximation for $\mathbb{A}$ and hence $\mathcal{P}$ defines an n^ε approximation for the MAX FA-INT problem. Thus the result follows. □

Finally, we consider the problem of finding a smallest set P_S of sets of states such that each set has a PDS (MinSetPDS).

Theorem 5.2. *The MinSetPDS problem is PSPACE-complete.*

Proof. We first prove that the problem is in PSPACE. Observe that in P_S we require at most one set for each pair of states of M and so if M has state set S then the set P_S of subsets has size at most $|S|(|S|-1)$. It is therefore sufficient to show that we can solve the problem of trying to find a set k of subsets where each subset corresponds to a PDS ($1 \leq k \leq |S|(|S|-1)$); if we can do this then a Turing Machine could start with $k = |S|(|S|-1)$ and then reduce k step by step until a set is found. Given k, a non-deterministic Turing Machine can thus initially guess such a set P_S of k subsets and for each such set $\bar{S} \in P_S$ the Turing Machine tries to build a PDS that distinguishes all of the states in $\bar{S}$. As before, for a given set $\bar{S}$ the process terminates when the upper bound on PDS length is exceeded or the PDS being built is sufficient. Since this can be performed in polynomial space we have that the result follows from Savitch's Theorem [61].

To see that the problem is PSPACE-hard it is sufficient to observe that M has a complete PDS if and only if it has a set P_S that satisfies the conditions of the MinSetPDS problem and contains only one set. The

[6]In [29], Gill presents this result on pg: 104, Theorem 4.3. Note that Gill named the set $\bar{S}$ as the *Admissible Set* i.e. the initial states of the underlying FSM.

result therefore follows from the complete PDS problem being PSPACE-hard [16]. □

6. INCOMPLETE ADAPTIVE DISTINGUISHING SEQUENCES

In some situations we want to use preset input sequences in testing and fault localisation since this requires a relatively simple test infrastructure: one that simply applies a sequence of inputs and observes the resultant outputs. However, efficiency can be improved if we use adaptive tests, where the next input to be applied is chosen on the basis of the observations made. In addition, it is known that the problem of deciding whether an FSM has a (complete) ADS can be solved in polynomial time and there is a polynomial upper bound on the size of such an ADS [16]. That is, we can decide in polynomial time whether the underlying FSM possesses a complete ADS and then decide whether to construct incomplete ADSs. Note that this flexibility exists for ADSs but not PDSs. In other words, in order to use incomplete PDSs, one may wish to first check the existence of a complete PDS. However we know that for both problems checking existence is PSPACE-complete [16]. These results, together with the complexity results in Section 5, provide the motivation for considering incomplete ADSs. In this section we therefore explore incomplete ADSs and report that the complexity results given for problems related to PDSs hold when we consider ADSs.

We assume that we are given a set $\mathbb{A} = \{A_1, A_2, \ldots, A_z\}$ of (minimal) finite automata with alphabet Σ and now describe the FSM $\mathcal{M}_2(\mathbb{A})$ that we construct. We mark the initial states of the finite automata so that the initial state of A_i is called 0_i and will let $\bar{S} = \{0_1, 0_2, \ldots, 0_z, Sink\}$ for a state $Sink$ described below and set 0_1 to be the initial state. We introduce a set $\mathcal{D} = \{d_1, d_2, \ldots, d_z\}$ of new inputs and so there exists one such input d_i for each $A_i \in \mathbb{A}$. The transitions of the finite automata from $\mathbb{A}$ with input alphabet Σ are inherited (and given output 0) and the remaining transitions are as follows

- $\delta(Sink, x) = Sink$ for all $x \in \Sigma \cup \mathcal{D}$.
- If $x \in \mathcal{D}$ then:
  - If $s \in F_i$ then $\delta(s, x) = s$; and
  - $\delta(s, x) = Sink$ otherwise.

The output function λ of $\mathcal{M}_2(\mathbb{A})$ is defined as follows in which $1 \leq i \leq z$.

$$\lambda(s,x) = \begin{cases} i, & \text{If } s \in F_i \text{ and } x = d_i, \\ 0 & \text{For all other cases,} \end{cases}$$

Unlike the previous reduction the output function does not enable us to recognise the states of finite automaton A_i while we are visiting the states in $Q_i \setminus F_i$. Instead, we can only distinguish states through applying an input from $\mathcal{D}$, possibly after a sequence of previous inputs. Further, we can only distinguish a state 0_i from $Sink$ through applying an input sequence w that takes A_i to an accepting state and then apply d_i. We now prove that we can construct an ADS for $\bar{S}$ if and only if the finite automata in $\mathbb{A}$ accept a common word.

In the following we represent an incomplete ADS for $\bar{S}$ by a set of input/output sequences: the input/output sequences produced from the states from $\bar{S}$.

Lemma 6.1. *Let us suppose that set $\mathbb{A} = \{A_1, A_2, \ldots, A_z\}$ of finite automata have a common alphabet Σ. The FSM $\mathcal{M}_2(\mathbb{A}) = (S, X, Y, \delta, \lambda, s_0)$ has an* ADS *for $\bar{S} = \{0_1, 0_2, \ldots, 0_z, Sink\}$ if and only if there is a word $w \in \Sigma^\star$ that is accepted by all of the finite automata (in which case input sequences wd_1, wd_1d_2, $wd_1d_2d_3$, $\ldots, wd_1d_2d_3 \ldots d_z$ define an* ADS*).*

Proof. We first show that if w is accepted by all the finite automata then input sequences wd_1, wd_1d_2, $wd_1d_2d_3, \ldots, wd_1d_2d_3 \ldots d_z$ define an ADS for $\bar{S}$. Since w is accepted by all the finite automata, input sequence w will take any initial state 0_i to an accepting state. We show that $wd_1d_2 \ldots d_j$ distinguishes states $0_i, 0_j$ for any $1 \leq i \neq j \leq z$. This follows from the fact that at state $\delta(0_j, w)$ the FSM will not change its state, will produce 0 when any input from set $\mathcal{D} \setminus \{d_j\}$ is applied, and will produce j as output if input d_j is applied. It is clear also that this word distinguishes any 0_j from $Sink$ since it will lead to the output j being produced from 0_j but not from $Sink$. Therefore, if there exists a word w that is accepted by all the finite automata, then FSM $\mathcal{M}_2(\mathbb{A})$ has an ADS for set $\bar{S}$ in the form of $wd_1, wd_1d_2, wd_1d_2d_3, \ldots, wd_1d_2d_3 \ldots d_z$.

Now assume that machine $\mathcal{M}_2(\mathbb{A})$ has an ADS for $\bar{S}$ and we are required to prove that there is some $w \in \Sigma$ in the intersections of the languages of the finite automata. Let us suppose that from $\bar{S}$ the ADS applies input sequence w and then input x such that the response to w does not distinguish any two elements of $\bar{S}$ but the response to wx distinguishes two or more states of $\bar{S}$. Then the input of x after w must lead to different outputs for two or more states in $\bar{S}$ and so we must have that $x \in \mathcal{D}$. If w does not take some 0_j to a final state then wx takes 0_j to state $Sink$ producing only zeros as output and so the ADS does not distinguish 0_j from $Sink \in \bar{S}$. Thus, w must take each A_i to a final state and so by definition we have that $w \in \Sigma^\star$ and w is in the languages defined by all of the A_i and so the result holds. □

We now show that we can check in PSPACE whether a set of states has an ADS.

Lemma 6.2. *Given deterministic, minimal and completely-specified FSM M and state set $\bar{S}$, the problem of deciding whether $\bar{S}$ has an* ADS *is in* PSPACE.

Proof. We will show that a non-deterministic Turing Machine can solve this using polynomial space. Such a machine will operate through a sequence of steps, extending the depth of the ADS by one in each step. It will maintain a set π of pairs of states and equivalence relation r as in the proof of Lemma 5.2 and again this uses polynomial space. As before, we start with $\pi = \{(s,s)|s \in \bar{S}\}$. In each step, if $(s,s') \in \pi$ then the current 'guess' takes s to s' and $(s,s') \in r$ if and only if the current 'guess' does not distinguish s and s'. A step involves the non-deterministic Turing Machine guessing a next input for each equivalence class of r and updating π and r accordingly. The machine also checks whether an ADS has been defined for $\bar{S}$: this is the case if and only if r relates no two different states of $\bar{S}$. Thus, if M has an ADS for $\bar{S}$ then this non-deterministic Turing Machine will find such an ADS using polynomial space.

Similar to before, we now have to consider the case where M does not have an ADS for $\bar{S}$ and require that the non-deterministic Turing Machine terminates. This is achieved by using the result that if M has n states and $\bar{S}$ has m states then M has an ADS for $\bar{S}$ if and only if it has such an ADS with length at most $\Sigma_{i=2}^{i=m} C\binom{n}{i} < 2^n$, where $C\binom{n}{i}$ is the number of ways of choosing a subset of size i of a set of size n [30][7]. The non-deterministic Turing Machine thus has a counter that gives the length of the current 'guess' and terminates with failure if the counter exceeds the upper bound. We require additional $O(\log_2(2^n)) = O(n)$ space for the counter and so the space required is polynomial.

The non-deterministic Turing Machine requires polynomial space in order to solve the problem and so the problem is in non-deterministic PSPACE; the result again follows from Savitch's Theorem [61]. □

The structure of $\mathcal{M}_2(\mathbb{A})$ ensures that when trying to distinguish states in $\bar{S}$ we gain nothing from adaptivity: once we have observed a non-zero output from one of the states we have distinguished this state from all other states in $\bar{S}$ (we must only observe zeros when starting in $Sink \in \bar{S}$). Thus, when exploring ADSs for $\bar{S}$ it is sufficient to consider input sequences.

We now show that the MaxSubSetADS problem, of finding an ADS that distinguishes the most states from some $\bar{S}$, is PSPACE-complete.

Theorem 6.1. *The* MaxSubSetADS *problem is* PSPACE-complete.

Proof. The problem being PSPACE-hard follows from Lemma 6.1 and the MAX FA-INT problem being PSPACE-hard. In order to see that the problem is in PSPACE, we prove that the following problem is in PSPACE: "*for a minimal, deterministic and completely-specified FSM with n states and $1 \leq k \leq n$, decide whether there is an* ADS *that distinguishes k states*". We can deduce that this problem is in PSPACE, by considering the algorithm presented in Lemma 6.2. This time as a preprocessing step the Turing Machine will guess a set of states $\bar{S}$ with cardinality k then the Turing Machine continues to implement the procedure that we describe in the proof of Lemma 6.2. Therefore the MaxSubSetADS problem is in PSPACE. □

[7]Theorem 1, bound (2).

Lemma 6.1 implies that the optimum solution to the MAX FA-INT problem constitutes an optimum solution to the MaxSubSetADS problem and hence we can reach the following conclusion.

Lemma 6.3. *Given a set $\mathbb{A}$ of finite automata, let $OPT_{\mathbb{A}}$ be the set of minimal words accepted by the maximum number of finite automata from $\mathbb{A}$. Further, let $\mathcal{M}_2(\mathbb{A})$ be the FSM constructed from $\mathbb{A}$ and also let $OPT_{\mathcal{M}_2(\mathbb{A})}$ be the set of minimal* ADSs *that maximise the size of the subset of $\bar{S}$ whose states are pairwise distinguished by* ADSs. *Then $w \in OPT_{\mathbb{A}}$ if and only if* ADS $wd_1, wd_1d_2, \ldots, wd_1 \ldots d_z$ *is in $OPT_{\mathbb{M}}$.*

Theorem 6.2. *There exists a constant $\varepsilon > 0$ such that approximating the* MaxSubSetADS *problem within ratio n^{ε} is* PSPACE-hard.

Proof. To prove that the problem of approximating MaxSubSetADS is PSPACE-hard, we consider an algorithm $\mathcal{P}$ that belongs to a complexity class $C < PSPACE$ and returns an n^{ε} approximation for the MaxSubSetADS Problem. In such a case, given a MAX FA-INT problem instance $\mathbb{A}$, we can construct FSM $\mathcal{M}_2(\mathbb{A})$ and using $\mathcal{P}$ we can obtain a solution $wd_1, wd_1d_2, \ldots, wd_1d_2 \ldots d_z$. But then Lemma 6.3 implies that w defines an approximation for $\mathbb{A}$ and hence $\mathcal{P}$ is also an approximation for the MAX FA-INT problem. The result thus follows. □

As with PDSs, in testing we might want a smallest set of ADSs that, between them, distinguish all states of M (MinSetADS).

Lemma 6.4. *The* MinSetADS *problem is in* PSPACE.

Proof. We can show that this problem is in PSPACE by following a procedure that is similar to the one we present in the proof of Theorem 6.1. As before, if the FSM M has state set S then P_S requires at most $|S|(|S|-1)$ sets. As a result, it is sufficient to prove that given k the following problem can be solved in PSPACE: is there a collection $P_S = \{\bar{S}_1, \bar{S}_2 \ldots \bar{S}_k\}$ of subsets of S such that for every pair s, s' of states there is some $\bar{S}_i$ such that $s, s' \in \bar{S}_i$ and for each $\bar{S}_i$ $(1 \leq i \leq k)$ there is an ADS that distinguishes the states of $\bar{S}_i$. The Turing Machine guesses such a P_S and then performs the remaining steps for each of the $\bar{S}_i$ separately. Clearly, this procedure takes polynomial space. The Turing Machine will return failure when it exceeds the bound given for the maximum depth, while if suitable ADSs are found then the Turing Machine returns success. □

In the proof of the following, given an instance of FA-INT problem $\mathbb{A} = \{A_1, A_2, \ldots, A_z\}$, we will define an

FSM $\mathcal{M}_3(\mathbb{A})$ that is the same as $\mathcal{M}_2(\mathbb{A})$ except for the following:

- For all $1 \leq i \leq z$ we add a state $0'_i$;
- We set $\bar{S} = \{0'_1, 0'_2, \ldots, 0'_z, Sink\}$;
- We introduce new input *st*; and
- We add the following transitions: from state $0'_i$ there is a transition to 0_i with label $st/0$ and all other inputs take $0'_i$ to $Sink$ with output 0. From all states other than the $0'_i$ the input of *st* leads to state $Sink$ and output 0.

The essential idea is that in order to distinguish two states from $\bar{S}$ an ADS must start with input *st* but this ensures that this ADS does not distinguish any two states from $S \setminus \bar{S}$ (and also does not distinguish any state in $S \setminus \bar{S}$ from $Sink$). Thus, any set of ADSs that distinguishes all of the states of $\mathcal{M}_3(\mathbb{A})$ can be partitioned into a subset that distinguishes the states of $\bar{S}$ and a subset that distinguish the states in $(S \setminus \bar{S}) \cup \{Sink\}$ and so there is an ADS for $\bar{S}$ if and only if a smallest set of ADSs for $\mathcal{M}_3(\mathbb{A})$ defines such an ADS.

Lemma 6.5. *The* MinSetADS *problem is* PSPACE-hard.

Proof. We again consider an instance $\mathbb{A} = \{A_1, A_2, \ldots, A_z\}$ of the FA-INT problem with a common alphabet Σ and we construct $\mathcal{M}_3(\mathbb{A})$. From Lemma 6.1 and the FA-INT problem being PSPACE-hard we know that the problem of deciding whether there is an ADS for $\{0_1, 0_2, \ldots, 0_z, Sink\}$ is PSPACE-hard. We will prove that any solution to the MinSetADS problem for $\mathcal{M}_3(\mathbb{A})$ also determines whether there is an ADS for $\{0_1, 0_2, \ldots, 0_z, Sink\}$.

Let us suppose that P_S is a smallest set of subsets of S such that for every pair of states s, s' with $s \neq s'$ there is some $\bar{S}' \in P_S$ that contains both s and s' and for every set $\bar{S}' \in P_S$ there is an ADS that distinguishes the states in $\bar{S}'$. By construction, any set $\bar{S}' \in P_S$ that contains $Sink$ and a state $s \in \bar{S} \setminus \{Sink\}$ must correspond to ADSs that start with *st*. Similarly, if $\bar{S}' \in P_S$ contains $Sink$ and some $s \in S \setminus \bar{S}$ then it must correspond to ADSs that do not start with *st*.

We will let P'_S denote the set of subsets of P_S that contain $Sink$ and at least one state $s \in \bar{S} \setminus \{Sink\}$. We will prove that there is an ADS that distinguishes all of the states of $\bar{S}$ if and only if P'_S contains only one set.

First assume that P'_S contains only one set. Thus, the one set in P'_S contains all states from $\bar{S}$ and this implies that there is an ADS for $\bar{S}$ as required.

Now assume that $\bar{S}$ has an ADS. By definition, no set in P'_S contains a state $s \notin \bar{S}$ and for all $s \notin \bar{S}$ we have that $P_S \setminus P'_S$ contain a set that has both s and $Sink$. Thus, for each $s, s' \in (S \setminus \bar{S}) \cup \{Sink\}$ with $s \neq s'$ we have that $P_S \setminus P'_S$ has a set that contains both s and s'. As a result, it is sufficient for the sets in P'_S to contain all pairs s, s' from $\bar{S}$ with $s \neq s'$. Since there is an ADS that achieves this, by the minimality of P_S we must have that P'_S contains only one set.

We now know that $\bar{S}$ has an ADS if and only if P'_S contains only one set and so if we can solve the MinSetADS problem for $\mathcal{M}_3(\mathbb{A})$ then we can decide whether $\bar{S}$ has an ADS. We can now note that $\bar{S}$ has an ADS if and only if the state set $\{0_1, 0_2, \ldots, 0_z, Sink\}$ of $\mathcal{M}_3(\mathbb{A})$ has an ADS: the ADS for $\bar{S}$ in $\mathcal{M}_3(\mathbb{A})$ starts with *st* and then applies an ADS for state set $\{0_1, 0_2, \ldots, 0_z, Sink\}$ of $\mathcal{M}_3(\mathbb{A})$. The result thus follows from Lemma 6.1 and the FA-INT problem being PSPACE-hard. □

We therefore have the following result.

Theorem 6.3. *The* MinSetADS *problem is* PSPACE-complete.

We saw that a fully distinguishing set $A = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k\}$ of ADSs defines the set $\{A(s_1), A(s_2), \ldots, A(s_n)\}$ of harmonised state identifiers. We also have the converse, that harmonised state identifiers can be used to construct a fully distinguishing set of ADSs, since each sequence in a state identifier defines an ADS (in which there is no adaptivity). Thus, the complexity results in this paper regarding ADSs correspond to equivalent results regarding harmonised state identifiers.

Given a set $\bar{S} \subseteq S$ and harmonised state identifiers $\{H_1, H_2, \ldots, H_n\}$, we can identify alternative subsets of the H_i that are sufficient to distinguish the states of $\bar{S}$. Let us suppose that $H'_i \subseteq H_i$ for all $s_i \in \bar{S}$. Then we will say that the H'_i form harmonised state identifiers for $\bar{S}$ if for all distinct $s_i, s_j \in \bar{S}$ we have sequences $w_i \in H'_i$ and $w_j \in H'_j$ such that a common prefix of w_i and w_j distinguishes s_i and s_j. The following shows how the MaxSubSetADS problem relates to problems regarding harmonised state identifiers.

Proposition 6.1. *Let us suppose that $\bar{S}$ is a set of states of deterministic, minimal and completely-specified FSM M. Then the states in $\bar{S}$ can be distinguished by a single* ADS *if and only if there exist harmonised state identifiers $H'_1, \ldots, H'_n$ for $\bar{S}$ where each H'_i contains only one input sequence.*

Proof. First assume that the states in $\bar{S}$ can be distinguished by a single ADS $\mathcal{A}$. Given state $s_i \in \bar{S}$ let H'_i denote the set containing one input sequence: the input portion of the input/output sequence produced when $\mathcal{A}$ is applied in state s_i. It is straightforward to check that the argument used in the proof of Proposition 3.1 applies and so the H'_i are state identifiers for $\bar{S}$ as required.

Now let us suppose that we have state identifiers $\{H'_i | s_i \in \bar{S}\}$ for $\bar{S}$ such that each H'_i contains only one input sequence, which we call w_i. Form a finite automaton $\mathcal{A}$ that is a tree with $|\bar{S}|$ leaves such that for each state $s_i \in \bar{S}$ the tree $\mathcal{A}$ has a path from the root to a leaf such that this path has label $w_i/\lambda(s_i, w_i)$. Since the H'_i define identifying sets for $\bar{S}$, for distinct states $s_i, s_j \in \bar{S}$ we have that a common prefix w of w_i, w_j distinguishes s_i and s_j. This ensures that the tree satisfies the required condition that from two distinct states s_i, s_j an ADS applies the same input sequence until these states are distinguished. Thus, $\mathcal{A}$ is an ADS for $\bar{S}$ as required. □

The following gives a relationship between an HSI problem and MinSetADS.

Proposition 6.2. *If the states in S can be distinguished by k ADSs then there are harmonised state identifiers $\{H_1, H_2, \ldots, H_n\}$ such that for all $s_i \in S$ we have that H_i has at most k input sequences.*

Proof. We will assume that the states in S can be distinguished by a set A of k ADSs. Given state $s_i \in \bar{S}$ let H_i denote the set containing the input portion of the input/output sequence produced when $\mathcal{A} \in A$ is applied in state s_i. By Proposition 3.1 the H_i are harmonised state identifiers for S as required. □

The last two results in this section give relationships between the results regarding ADSs and corresponding problems regarding harmonised state identifiers. While the focus of this paper is on PDSs and ADSs, the results given here (and techniques used to prove them) might provide additional insights into optimisation problems for harmonised state identifiers.

7. EMPIRICAL STUDY

In this section, we first present a greedy algorithm that aims to compute a fully distinguishing set with minimum cardinality. Later we present the results of experiments using randomly generated FSMs and some benchmark FSMs.

The aim of the experiments was to compare the state identifiers constructed using the following approaches:

1. Using the standard approach for generating a characterisation set [28]: we call this the *W-generation method* (W).
2. Using the standard approach for generating harmonised state identifiers [19, Algorithm 2 in Appendix 2]: we call this the *HSI-generation method.*
3. Using a proposed greedy algorithm to generate a fully distinguishing set of ADSs and deriving state identifiers from these: we call this the *ADS-method.*

In the experiments we compared: 1) the number of input sequences per state; and 2) the mean length of state identifiers. We also report on the results of an experiment that investigated the time taken by the greedy algorithm.

7.1. Greedy Algorithm

Before the algorithm is presented, we first define notation used. We present the list of symbols with their definitions in Table 2. The greedy algorithm receives an FSM M and integer ℓ and it returns a set of trees $\mathbf{T} = \{T_1, T_2, \ldots\}$ such that all trees in this set have depth at most ℓ and set $\mathbf{T}$ defines a fully distinguishing set (if such a $\mathbf{T}$ exists).

Symbol	Description
$\mathbf{T}$	A set of tree structures.
T	A tree structure.
N, E	Set of nodes, set of edges.
$\mathcal{I}(v)$,$\mathcal{C}(v)$	Initial and Current sets for node v.
$i(v), o(v)$	Input sequence, output sequence for node v.
$\mathcal{M}$	A set of current sets.
$\mathcal{N}$	Set of set of nodes used by the Greedy algorithm.
$\ell \in \mathbb{Z}_{\geq 1}$	Upper bound on the tree height.
$\mathcal{Q}$	A set of pairs of states.
$\Phi_x(\mathcal{Q}, \mathcal{N}_x) \in \mathbb{R}_{\geq 0}$	Heuristic function 1.
$\Theta_x(\mathcal{M}, \mathcal{N}_x) \in \mathbb{Z}_{\geq 0}$	Heuristic function 2.
$\mathcal{F} : S \times S \to \in \{0,1\}$	A function used by the Heuristic function 2.
$\text{argmax}_{\{\cdots\}} f(.)$	A function that returns values for a set of variables $\{\cdots\}$ such that f is maximised.
$\text{argmin}_{\{\cdots\}} f(.)$	A function that returns values for a set of variables $\{\cdots\}$ such that f is minimised.

TABLE 2: Nomenclature for the greedy algorithm.

The following provides a brief summary of the algorithm (the details are given later). The greedy algorithm starts with an empty tree and adds new leaves in an iterative manner. Given a leaf v the algorithm has a heuristic regarding how to choose an input to apply in v and this essentially operates by maximising the number of states distinguished. This process continues until either all states have been distinguished or the preset maximum depth ℓ has been reached. In the former case the algorithm terminates and in the latter the set of states distinguished is updated and the process is repeated.

7.1.1. Basic Notation

Given set B of states, $B_{x/y}$ will denote the subset of B such that each state in $B_{x/y}$ produces output y when input x is applied. Thus, $B_{x/y} = \{s \in B | y = \lambda(s, x)\}$. Similarly, we let $\hat{B}_{x/y}$ denote the states reached from states in set $B_{x/y}$ when input symbol x is applied. Thus, $\hat{B}_{x/y} = \{\delta(s, x) | s \in B_{x/y}\}$.

A tree $T(E, N) \in \mathbf{T}$ consists of a set of edges (E) and nodes (N). An edge $e \in E$ is labeled with an input output pair x/y where $x \in X$ and $y \in Y$. A node $v \in N$ captures the following information: Strings $i(v)$ and $o(v)$ that give the input and output sequences that label the path from the root of $T(E, N)$ to the node v, the initial set $\mathcal{I}(v)$, and the current set $\mathcal{C}(v)$. The initial and the current sets are defined as follows: $\mathcal{I}(v) = \{s \in S | o(v) = \lambda(s, i(v))\}$ and $\mathcal{C}(v) = \{\delta(s, i(v)) | s \in \mathcal{I}(v)\}$. We say that input x *refines* a node v if the states in the current set of node v do not produce the same output symbol when input x is applied i.e. x refines v if there exist $s, s' \in \mathcal{C}(v)$ such that $\lambda(s, x) \neq \lambda(s', x)$.

For the root node v_1 we have that $i(v_1) = o(v_1) = \varepsilon$ and $\mathcal{I}(v_1) = \mathcal{C}(v_1) = S$. Input sequence $i(v')$ is defined as $i(v') = i(v)x$ where v is the parent node of the current node v' and x is the input retrieved from the edge between v and v'. Further, $o(v') = o(v)y$ where v is the parent node of the current node v' and y is the output retrieved from the edge between v and v'. In T there are two types of nodes: a node is a *leaf node* if and only if it has no outgoing edges; otherwise it is an *internal node.*

7.1.2. The Algorithm

The greedy algorithm receives an FSM M and positive integer ℓ. The summary of the Greedy Algorithm is given in Algorithm 1. Initially $\mathcal{Q}$ contains the set of all pairs of distinct states. The algorithm iterates until the set $\mathcal{Q}$ becomes empty (Line 3). At each iteration, the greedy algorithm forms a tree structure T by introducing a root node v_1 (Lines 4–5). The root node has the following information: $\mathcal{I}(v_1) = S$, $\mathcal{C}(v_1) = S$, $i(v_1) = \varepsilon$, $o(v_1) = \varepsilon$.

The greedy algorithm constructs a tree T iteratively and at each iteration a single node *var* is handled.

For a given node *var* for each input $x \in X$ and for each $\mathcal{C}(var)_{x/y}$ in set $\{\mathcal{C}(var)_{x/y} | y \in \lambda(\mathcal{C}(var), x)\}$ it introduces a new node v such that $\mathcal{C}(v) = \hat{\mathcal{C}}(var)_{x/y}$, $\mathcal{I}(v) = \{s \in \mathcal{I}(var) | \lambda(\delta(s, i(var), x)) = y\}$, $i(v) = i(var)x$ and $o(v) = o(var)y$ (Lines 10 – 15)

The proposed algorithm uses a set of sets of nodes $\mathcal{N} = \{N_1, N_2, \ldots, N_{|X|}\}$ for inspecting the suitability of input symbol x as follows: After forming new nodes (i.e. v's) the algorithm forms set N_x and adds these nodes to the set $\mathcal{N}$ (Line 16). Afterwards it checks whether $\mathcal{N}$ is a subset of $\mathcal{M}$. The set $\mathcal{M}$ holds the set of current sets that belong to nodes which cannot be refined. Therefore, if the current sets of all possible children of the current node are in set $\mathcal{M}$, there is no point in investigating this node any more, consequently, we also add the current set of such a node to $\mathcal{M}$ as well (Lines 17 – 18).

Otherwise the greedy algorithm evaluates the "goodness" of inputs by calling (Lines 19 – 20) a heuristic function which is defined as follows:

$$\Phi_x(\mathcal{Q}, N_x) = \sum_{v \neq v' \in N_x} |\mathcal{Q} \cap \mathcal{I}(v) \times \mathcal{I}(v')| \qquad (3)$$

Algorithm 1: Greedy Algorithm.

```
   Data: FSM $M$, $\ell$
   Result: A set of trees $\mathbf{T}$
   begin
1      Let $\mathcal{Q} \leftarrow \{(s_1, s_2), (s_1, s_3), \ldots (s_{n-1}, s_n)\}$ be a
       set of distinct pairs of states.
2      Set $\mathbf{T} \leftarrow \emptyset$ and $\mathcal{M} \leftarrow \emptyset$.
3      while $\mathcal{Q} \neq \emptyset$ do
4          $v_1 \leftarrow (S, S, \varepsilon, \varepsilon)$, $\mathcal{N} \leftarrow \emptyset$,$a \leftarrow 0$, $h \leftarrow 0$
5          Add $v_1$ to tree $T(N, E)$, $var \leftarrow v_1$
6          while $var \neq NULL$ do
7              $max \leftarrow 0$, $index \leftarrow -1$
8              if $|x(var)| < \ell$ then
9                  for $x \in X$ do
10                     foreach $y \in \lambda(\mathcal{C}(var), x)$ do
11                         Generate new node $v$
12                         $\mathcal{I}(v) \leftarrow \{s \in \mathcal{I}(var) | \lambda(\delta(s, i(var), x)) = y\}$
13                         $\mathcal{C}(v) \leftarrow \hat{\mathcal{C}}(var)_{x/y}$
14                         $i(v) \leftarrow i(var)x$
15                         $o(v) \leftarrow o(var)y$
16                         Add $v$ to $N_x$
17                 if $\mathcal{N} \subseteq \mathcal{M}$ then
18                     Add $\mathcal{C}(var)$ to $\mathcal{M}$
19                 else if $\forall x \in X, \Phi_x(\mathcal{Q}, N_x) = 0$ then
20                     $index \leftarrow \mathrm{argmin}_{x \in X}\, \Theta_x(\mathcal{M}, N_x)$
21                 else
22                     $index \leftarrow \mathrm{argmax}_{x \in X}\, \Phi_x(\mathcal{Q}, N_x)$
23                 if $index = -1$ then
24                     Add $\mathcal{C}(var)$ to $\mathcal{M}$
25                 else
26                     for $v \in N_{index}$ do
27                         if No proper ancestor of node var have a current set $\mathcal{C}(v)$
                           then
28                             Add node $v$ to $N$ and add edge to $E$.
29             $var \leftarrow$ next_unvisited_node
30         for All pair of leaf nodes $v, v'$ where $v \neq v'$
           do
31             if $s \in \mathcal{I}(v) \wedge s' \in \mathcal{I}(v')$ then
32                 Pop pair of states $(s, s')$ from $\mathcal{Q}$ and
                   push $T$ onto $\mathbf{T}$ once.
33     Return $\mathbf{T}$
```

For any pair of nodes Heuristic 3 forms a set of pairs of states and counts the number of occurrences of pairs in set $\mathcal{Q}$. That is to say Heuristic 3 will return the number of pairs of states in $\mathcal{Q}$ distinguished. Intuitively a "good" input x maximises this mass function: for all $x' \in X, x \neq x'$ we have that $\Phi_{x'} \leq \Phi_x$.

Now consider the machine M_2 in Figure 6. According to Heuristic 3, the greedy algorithm will initially select input a to distinguish state s_3 from other states. Afterwards, the algorithm will try to distinguish states s_1, s_2 and s_4. However, according to Heuristic 3, there is no difference between inputs a and b and thus, the

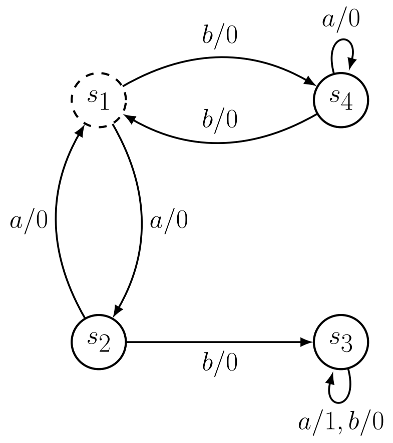

FIGURE 6: An FSM M_2 with initial state s_1.

greedy algorithm can try input a repeatedly and fall into a loop. To prevent this, in such cases, (i.e. if Heuristic function 3 cannot differentiate between inputs), the greedy algorithm decides the next input by the usage of the following greedy function (Lines 19 – 20): Let $\mathcal{M} = \{\mathcal{C}(1), \mathcal{C}(2), \ldots, \mathcal{C}(|\mathcal{M}|)\}$,

$$\Theta_x(\mathcal{M}, N_x) = \sum_{v \in N_x, \mathcal{C}(i) \in \mathcal{M}} \mathcal{F}(\mathcal{C}(i), \mathcal{C}(v)) \qquad (4)$$

where $\mathcal{F}$ is a binary function which returns 1 if and only if the parameters $\mathcal{C}(i)$ and $\mathcal{C}(v)$ are identical sets. Otherwise it returns 0. Function $\mathcal{F}$ is defined as follows:

$$\mathcal{F}(\mathcal{C}(v), \mathcal{C}(v')) = \begin{cases} 1 & \text{if } \mathcal{C}(v) = \mathcal{C}(v') \\ 0 & \text{Otherwise} \end{cases} \qquad (5)$$

If the node cannot be refined by an input, the greedy algorithm declares a failure and adds the current set of this node to set $\mathcal{M}$. Otherwise, the greedy algorithm adds the nodes and edges that are obtained by the corresponding input to the current tree T. While doing this the greedy algorithm checks whether the current set of the new node exists in one of its proper ancestor v' i.e. $\exists v' \in N$ such that $\mathcal{C}(v) = \mathcal{C}(v')$ and there exists a simple path from v' to v (Lines 25 – 27).

Afterwards the greedy algorithm selects another unvisited node and repeats the procedure (Line 29). The greedy algorithm repeatedly executes this scheme until each node is processed or the depth of the tree T becomes larger than ℓ.

The greedy algorithm removes a pair of states (s, s') from $\mathcal{Q}$ if s, s' are members of initial sets of different leaf nodes i.e. $s \in \mathcal{I}(v) \wedge s' \in \mathcal{I}(v')$ for $v \neq v'$ and the tree is pushed onto $\mathbf{T}$ (Lines 30 – 32). Finally, if $|Q| = 0$ the algorithm terminates and returns set $\mathcal{Q}$ (Line 33). Since the greedy algorithm is a heuristic the resultant tree T need not be optimal; later we report the results of experiments used to explore the effectiveness of this approach.

We now need to show that at each iteration the greedy algorithm computes an incomplete ADS. In order to achieve this we first need to emphasise some properties of tree T. First recall that the greedy algorithm selects a single set of nodes N_x while constructing a tree T and since N_x is constructed by a single input x, the outgoing edges are labeled by identical inputs and are labeled with different outputs. Therefore the following immediately follows from the construction of tree T.

Corollary 7.1. *Let v be an internal node of tree T with children $v_1, v_2, \ldots, v_p$ and let x be the input portion of the labels of the edges from node v. The following hold:*

1. *$\delta(\mathcal{C}(v), x) = \cup_{i=1}^{p} \mathcal{C}(v_i)$.*
2. *For all $1 \leq i \leq p$ we have that $|\lambda(\mathcal{I}(v_i), x(v_i))| = 1$.*
3. *For all $1 \leq i < j \leq p$ we have that $\lambda(\mathcal{I}(v_i), x(v)) = \lambda(\mathcal{I}(v_j), x(v))$ and $\lambda(\mathcal{I}(v_i), x(v)x) \neq \lambda(\mathcal{I}(v_j), x(v)x)$.*

Moreover, consider distinct leaf nodes (v, v') then using Corollary 7.1 we know that the output observed from any pair of states $s \in \mathcal{I}(v)$ and $s' \in \mathcal{I}(v')$ are different.

Corollary 7.2. *Let v, v' be distinct leaf nodes of tree T. If $s \in \mathcal{I}(v)$ and $s' \in \mathcal{I}(v')$ then $\lambda(s, x(v)) \neq \lambda(s', x(v))$ and $\lambda(s, x(v')) \neq \lambda(s', x(v'))$.*

Now we show that a tree T returned by the greedy algorithm defines an incomplete ADS.

Lemma 7.1. *Let T be a tree returned by the greedy algorithm such that $\bar{N} = \{v_1, v_2, \ldots, v_p\}$ is the set of leaf nodes of T. Let $\bar{S}$ be a set of states such that for all $1 \leq i \leq p$ we have that $|\mathcal{I}(v_i) \cap \bar{S}| \leq 1$. Then T defines an incomplete* ADS *for set $\bar{S}$.*

Proof. We will show that T can be used to construct an incomplete ADS $\mathcal{A}$ for $\bar{S}$. Take a copy of tree T and for every node v remove from $\mathcal{I}(v)$ all states not in $\bar{S}$. Now remove all nodes with empty initial sets to form $\mathcal{A}$.

Now we need to show that $\mathcal{A}$ is an incomplete ADS for $\bar{S}$. By the construction of $\mathcal{A}$, each leaf node must be labeled by a singleton set. To see this, assume that an initial set of a leaf node contains two or more states. Since we drop states that are not in $\bar{S}$ this implies that there exist distinct $s, s' \in \bar{S}$ such that $s, s' \in \mathcal{I}(v_a)$ for some a, providing a contradiction.

Moreover, it is easy to see that each internal node is labeled by a set of states, and each edge is labeled by an input output pair. Further, for an internal node in $\mathcal{A}$ there are at most $|Y|$ outgoing edges such that edges from a common node have identical input labels and different output labels. Thus, using Corollary 7.1 and Corollary 7.2 we can deduce that conditions of being an incomplete ADS given in Definition 3.3 are satisfied.

Therefore T defines an incomplete ADS for $\bar{S}$. □

Although the algorithm is easy to implement, it may not compute a fully distinguishing set for a given FSM. This will happen if the upper bound on ADS length is too short. Note that for an FSM M with n states, every pair of states is distinguished by a sequence of length at most $n-1$ and it is sufficient to use at most $n-1$ such sequences in order to distinguish all of the states of M. Thus, the algorithm is guaranteed to return a fully distinguishing set if we use a value of ℓ that is $n-1$ or larger.

Now consider the complexity of the greedy algorithm. First we can observe that the algorithm may introduce 2^n-1 (excluding the empty set) nodes to the set $\mathcal{M}$ therefore in the worst case the loop controlled by Line 3 can executes exponentially many times. However, if we assume that the algorithm manages to introduce an incomplete ADS on each iteration then, due to the cardinality of set $\mathcal{Q}$, the loop will iterate at most n^2 times. In fact, we can improve on this analysis as follows. Let us suppose that we have added k ADSs and consider the equivalence relation $\sim_k$ on the set S of states defined by two states being equivalent if and only if none of these k ADSs distinguish them. Then, $\sim_k$ must have at least $k+1$ equivalence classes since each time we add another ADS, this ADS must increase the number of equivalence classes by at least 1. Thus, since a fully distinguishing set of ADSs defines an equivalence relation with n equivalence classes, there can be at most $n-1$ iterations of the outer loop. The loop on Line 6 iterates at most ℓ times and the loop on Line 9 iterates $|X|$ times and in the worst case at each iteration it applies input x to n states (Line 10). The loop on Line 26 iterates n times and at each iteration it requires n steps of computation. The last loop on Line 30 iterates n^2 times therefore the complexity is $O(n\ell(mn+n^2))$ if each iteration is successful in finding an ADS that distinguishes two or more states not previously distinguished. Later we report on the results of experiments that explored the time taken to generate a set of incomplete ADSs.

Now let us consider the FSM given in Figure 2. The algorithm initialises $\mathcal{Q} = \{(s_1,s_2),(s_1,s_3),(s_1,s_4),(s_2,s_3),(s_2,s_4),(s_3,s_4)\}$ and node var where $\mathcal{I}(var) = \mathcal{C}(var) = \{s_1,s_2,s_3,s_4\}$. Then, the algorithm produces a set of nodes from node var with input symbols $X = \{a,b,c\}$. Since $(\Phi_a(Q,N_a) = \Phi_b(Q,N_b) = \Phi_c(Q,N_c)) > 1$, the algorithm executes the instruction given at Line 19 of Algorithm 1 and selects the input a. Thus, new nodes v_1 and v_2 are added to N where $\mathcal{C}(v_1) = \{s_1\}$, $\mathcal{I}(v_1) = \{s_2\}$ and $\mathcal{C}(v_2) = \{s_3,s_4,s_4\}$, $\mathcal{I}(v_2) = \{s_1,s_3,s_4\}$ and corresponding edges are added. In the second iteration node v_2 is selected (we omit v_1) and again the new nodes are retrieved and since for inputs b,c we have that $\Phi_b(Q,N_b) = \Phi_c(Q,N_c)) > 0$ the algorithm selects input b in Line 19. With input b new nodes v_3 and v_4 are generated where $\mathcal{C}(v_3) = \{s_2\}$, $\mathcal{I}(v_3) = \{s_1\}$ and $\mathcal{C}(v_4) = \{s_1,s_1\}$, $\mathcal{I}(v_4) = \{s_3,s_4\}$. With the remaining node (v_4), the algorithm cannot proceed further. The algorithm therefore removes pairs $(s_1,s_2),(s_1,s_3),(s_1,s_4),(s_2,s_3),(s_2,s_4)$ from $\mathcal{Q}$ and loops. Note that the remaining pair in set $\mathcal{Q}$ is (s_3,s_4) and the algorithm selects input b and processes the selection. Therefore the state identifiers given by the algorithm are as follows: $H_1(A) = \{ab\}$, $H_2(A) = \{a\}$, $H_3(A) = \{ab,b\}$, and $H_4(A) = \{ab,b\}$.

Note that Lines 20 and 22 introduce nondeterminism, that is when the cardinalities of sets returned by argmin or argmax are greater than one, the algorithm selects an input according to some order dictated by the implementation (e.g. in lexicographic order). Therefore the proposed algorithm may also compute the following harmonised state identifiers: $H_1(A) = \{a,cb\}$, $H_2(A) = \{a,cb\}$, $H_3(A) = \{cb\}$, and $H_4(A) = \{c\}$.

In the next subsection we present results of experiments that evaluated the use of incomplete ADSs.

7.2. Experimental evaluation

7.2.1. DS Generation and Evaluation

This section describes experiments used to explore the performance of the greedy algorithm by evaluating the state identifiers derived from the resultant ADSs. We randomly generated FSMs with 4, 6, and 8 inputs and outputs using the tool utilised in [14, 63].

The FSMs were constructed as follows: First, for each input x and state s_i we randomly assigned the values of $\delta(s_i,x)$ and $\lambda(s_i,x)$. After an FSM M was generated we checked its suitability as follows. We checked whether M is strongly connected[8]. Afterwards we checked that M is minimal and then used the LY-algorithm [16] to check that M does not have a complete ADS. If the FSM failed one or more of these tests then we omitted this FSM and produced another. Consequently, all FSMs were strongly connected and minimal, and had no complete ADS.

By following this procedure we constructed 200 FSMs with 5 states, 200 FSMs with 10 states, ..., 200 FSMs with 100 states. This was done for each size of the input and output alphabets so in total we used $1.2*10^4$ FSMs. We used an Intel Xeon E5-1650 CPU at 3.20GHz with 16 GB RAM to carry out these tests. We implemented the three methods, for generating state identifiers, using C++ and compiled on Visual Studio .Net 2013. The W-generation and HSI-generation methods were implemented according to the descriptions presented in references [28, 19]. As explained earlier, for each FSM we also used the greedy algorithm to construct a fully distinguishing set of ADSs and generated state identifiers from these.

Since the outputs are uniformly distributed during the generation of FSMs, one would expect the average

[8] M is strongly connected if for any pair (s,s') of states of M there is some input sequence that takes M from s to s'.

depth of the ADSs to be around $\lceil \log_q n \rceil$, where n and q are the number of states and the number of outputs, respectively. For our experiments with 4, 6 and 8 outputs and the number of states ranging between 5 and 100, the length of ℓ is expected to be 2–4 for 4 outputs ($\lceil \log_4 5 \rceil = 2$ and $\lceil \log_4 100 \rceil = 4$), 1–3 for 6 outputs ($\lceil \log_6 5 \rceil = 1$ and $\lceil \log_6 100 \rceil = 3$), and 1–3 for 8 outputs ($\lceil \log_8 5 \rceil = 1$ and $\lceil \log_8 100 \rceil = 3$). For each FSM, we set the upper bound on ADS depth to be twice this value $\lceil \log_q n \rceil$ i.e. $\ell = 2 * \lceil \log_q n \rceil$. With these values, we were able to produce fully distinguishing sets.

We present the results using boxplot diagrams generated by the ggplot2 library of the tool R [64, 65, 66]. For each box the first quartile corresponds to the lowest 25% of data, the second quartile gives the median, and the third quartile corresponds to the highest 25%. For each boxplot we added the smoothing line computed with the LOESS [67] method, and the semi-transparent ribbon surrounding the solid line is the 95% confidence interval. We used functions $\mathcal{SI}$ and $\mathcal{LI}$ described in Section 4 to evaluate experiment results.

7.2.2. *Number of Input Sequences per State*

We summarise this study in Figures 7, 8 and 9, where $p/q = 4/4, 6/6$ and $8/8$ respectively.

In Figure 7, we observe that the boxplot indicates that the average number of input sequences per state are comparable for *HSI* and *ADS*. Besides, as expected, the number of input sequences per state is high when W is used. Moreover, we observe that the results of the $\mathcal{SI}$ metric increase with the number of states. In Figure 8 we see that when $n \geq 30$ the results of $\mathcal{SI}$ are lower when ADSs are used. Similar observation can be made in Figure 9 when $n \geq 25$.

To support our observations, we used R to perform a non-parametric *Kruskal-Wallis Significance* [68] test on the HSI and ADS results. For each method (HSI, ADS), for each state number (n) and for each input/output value (p/q), we constructed two sets of samples such that one set holds the $\mathcal{SI}$ results for ADS and the other set holds the $\mathcal{SI}$ results for HSI. Afterwards, we ran the Kruskal-Wallis difference test on these sets of samples. The *null hypothesis* ($H0$) assumes that these two sets of samples have identical distributions. We selected the α value to be 0.05 and $df = 1$[9]. Therefore according to the table given for the Chi-Squared values in [69], if the null-hypothesis is correct then the Chi-Squared values ($\mathcal{X}^2$) of these measurements should be smaller than 3.841. Otherwise, we should reject the null-hypothesis and suggest that there is a significant difference.

The results are given in Table 3. We observe that in all cases we reject the null–hypothesis. Furthermore, it seems that the cardinalities of the sets of inputs and outputs have an impact on the sizes of the harmonised state identifiers constructed by HSI and ADS. Based on the experimental studies, it appears that using a fully distinguishing set of ADSs can reduce the number of state identifiers. We did not use statistical tests to compare W and ADS since the HSI-generation method outperformed the W-generation method.

[9]Here *df* stands for the *Degree of Freedom*, which is given by $k - 1$ where k is the number of samples supplied to the Kruskal-Wallis test and in our case $k = 2$.

7.2.3. *Length of Input Sequences*

The results of the experiment are given in Figures 10, 11 and 12 where input output numbers are $4, 6$ and 8 respectively.

Overall, the results suggest that the use of a fully distinguishing set reduces the length of the state identifiers and the difference between the results of ADS and the HSI increases with the number of states.

Furthermore we observe that the average length of the state identifiers constructed by the W-generation and the HSI-generation methods are comparable.

We again applied the Kruskal Wallis test on the results. For each method (HSI, ADS), for each state number (n) and for each input/output value (p/q), we constructed two sets of samples such that one set holds the $\mathcal{LI}$ results for the ADS and the other set holds the $\mathcal{LI}$ results for the HSI. Afterwards, we ran the Kruskal-Wallis difference test on these sets of samples. The results are given in Table 4. The results suggest that average lengths of state identifiers are statistically different as the results reject the null hypothesis.

7.2.4. *The effect of the size of input output symbols*

In this section we investigate the effect of the size of input output symbols on the quality of the state identification sequences. Figure 13 shows how the $\mathcal{SI}$ values varies as a function of the number of inputs.

We observed that the $\mathcal{SI}$ values reduce as the number of inputs increases. This is as expected since with more input symbols there are more opportunities to construct sequences that are capable of distinguishing more states. This is validated with the relation between the reduction ratio and the number of states; the reduction ration increase as the number of states increases.

We also investigate the variation of $\mathcal{LI}$ values with respect to the number of input and output symbols. The results are given in Figure 14. We can make a similar observation for the length of the state identifiers. That is as the number of input output symbols increase, the $\mathcal{LI}$ values reduces.

7.2.5. *Case Studies*

While using randomly generated FSMs allowed us to perform experiments with many subjects and so apply statistical tests, it is possible that FSMs used in practice differ from these randomly generated FSMs. We therefore decided to complement the experiments with some case studies. In this subsection we present the results of experiments conducted on FSM specifications retrieved from the ACM/SIGDA benchmarks, a set

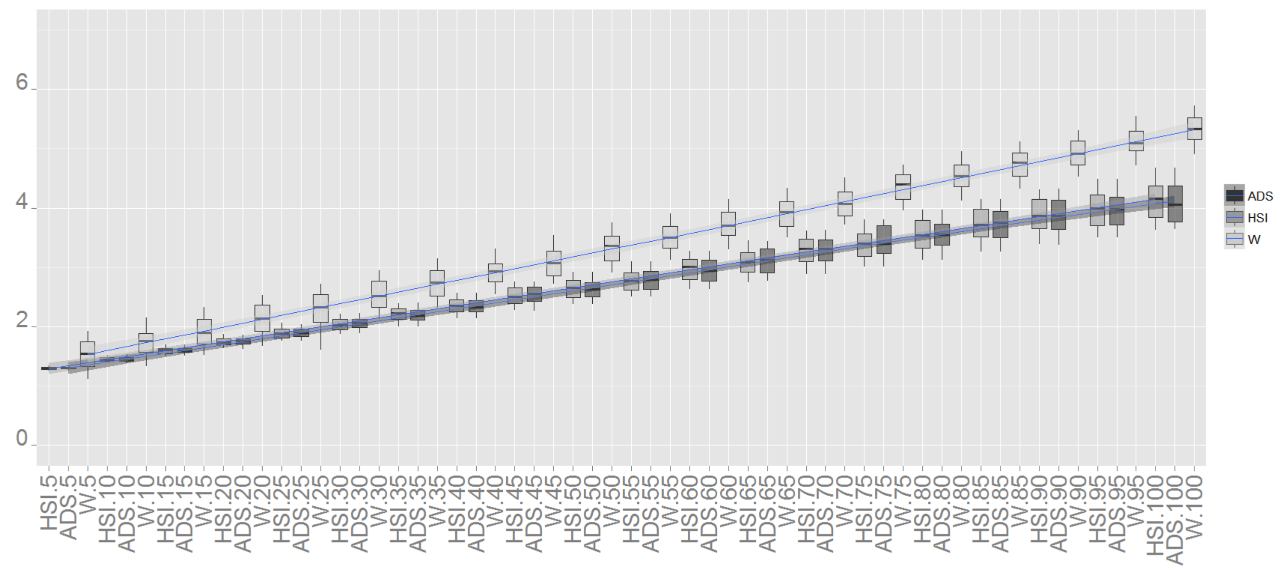

FIGURE 7: Comparison of average number of input sequences per state. Each boxplot summarises the distributions of 200 FSMs where $p = 4, q = 4$.

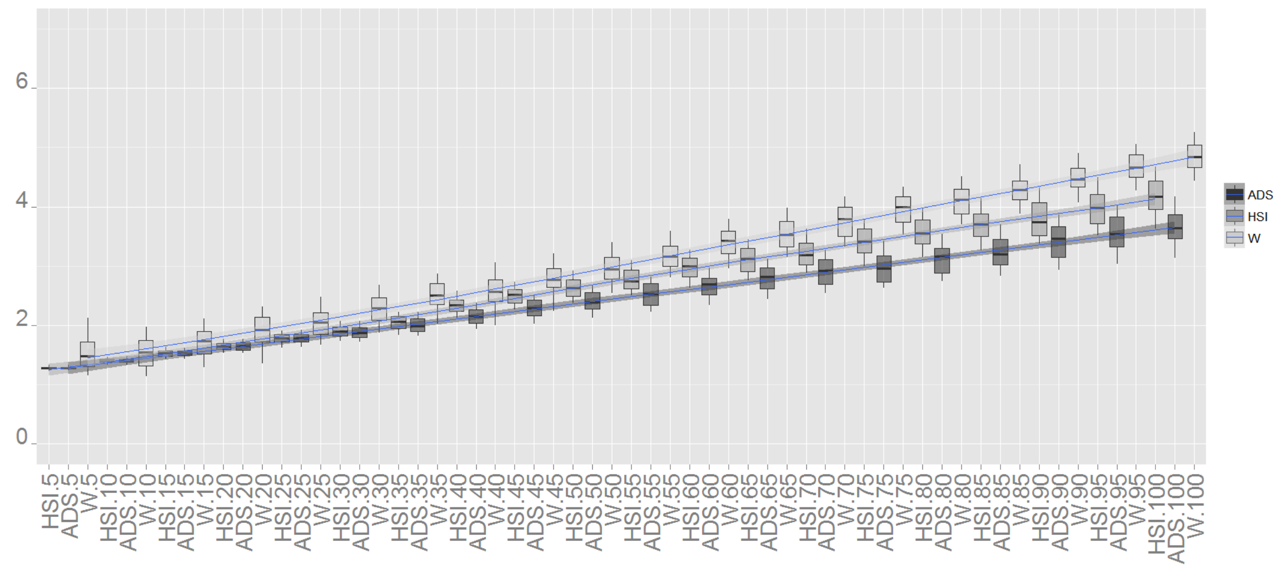

FIGURE 8: Comparison of average number of input sequences per state. Each boxplot summarises the distributions of 200 FSMs where $p = 6, q = 6$.

of test suites (FSMs) used in workshops between 1989-1993 [70]. The benchmark suite has 59 FSM specifications ranging from simple circuits to more advanced circuits obtained from industry. The FSM specifications are presented in the *kiss2* format. In order to process FSMs, we converted the *kiss2* file format to our FSM specification format. We only used FSMs from the benchmark that were minimal, deterministic, had no complete ADS, and had fewer than 10 input bits[10]. 19% of the FSMs had more than 10 input bits, 15% of the FSMs had complete ADS, and 38% were not minimal. 31% of the FSM specifications passed all of the tests. We computed state identifiers using the W-generation, HSI-generation and ADS methods and in Table 5 we present the results. We observe that the results, except for the FSM s386, are similar to those obtained in the experiments carried out with randomly generated FSMs.

7.2.6. *Time Comparison*

The average time required to compute state identifiers for randomly generated FSMs with different methods are provided in Figure 15. The results have

[10]Since the circuits receive inputs in bits, and since n bits correspond to 2^n inputs, we do not consider FSMs with $n \geq 10$ bits

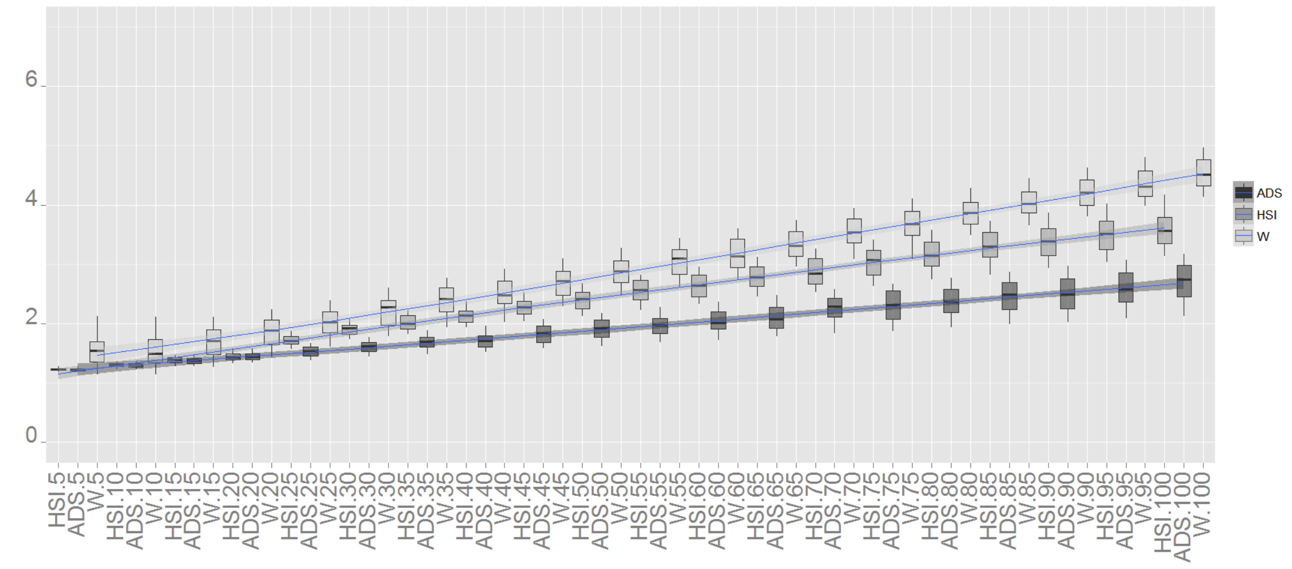

FIGURE 9: Comparison of average number of input sequences per state. Each boxplot summarises the distributions of 200 FSMs where $p = 8, q = 8$.

[t]

Input/output (p/q) values	Corresponding $\mathcal{X}^2$–values for different number of states (n). Reject H_0 when $\mathcal{X}^2 > 3.841$																			
	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
4/4	29.56	57.87	58.2	54.83	43.47	45.58	38.41	69.97	33.02	48.72	80.35	25.36	49.53	78.7	33.66	67.73	60.85	86.54	78.46	82.07
6/6	68.28	69.96	40.26	61.57	25.39	49.71	50.26	95	40.97	39.59	72.45	91.33	54.15	56.04	27.71	24.09	78.14	31.12	94.48	52.35
8/8	68.16	87.28	52.21	83.74	71.85	43.54	68.69	88.82	58.12	64.03	36.7	64.4	48.98	83.11	65.31	49.45	64.52	62.51	22.02	21.59

TABLE 3: The results of a Kruskal-Wallis Significance Tests performed on average length of the state identifiers per state.

important implications. Although the W-generation method, HSI-generation method and ADS-method have different ways of computing a state identifiers, with these settings, the times required to construct state identification sequences were comparable. Moreover as expected, the time grew slowly with the number of states and with the number of inputs. This suggests that all three methods will scale well as the number of states of an FSM increases.

The results for the case studies are presented in figure 16. These results are similar to those for randomly generated FSMs except for specifications *sand* and *nucpwr*. The specifications *sand* and *nucpwr* have relatively high numbers of inputs and this may well have affected the computation time. Interestingly, the ADS method took slightly longer than the other methods for these two case studies.

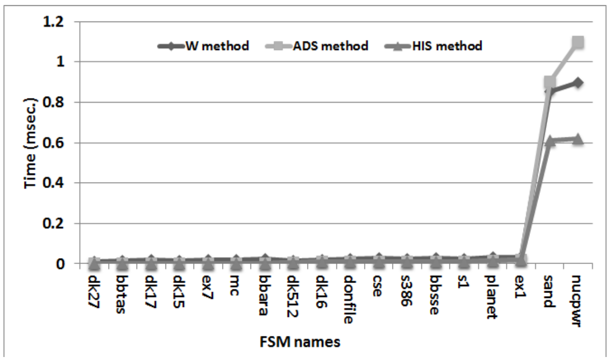

FIGURE 16: Comparison of average time to construct state identification sequences for case studies.

8. CONCLUSIONS

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 tests from either finite state machines (FSMs) or labelled transition systems. Many automated FSM based test techniques use complete distinguishing sequences (DSs) to check the state of the system under test after a transition. While complete DSs have many desirable properties, an FSM M need not have a complete DS. However, we might still have

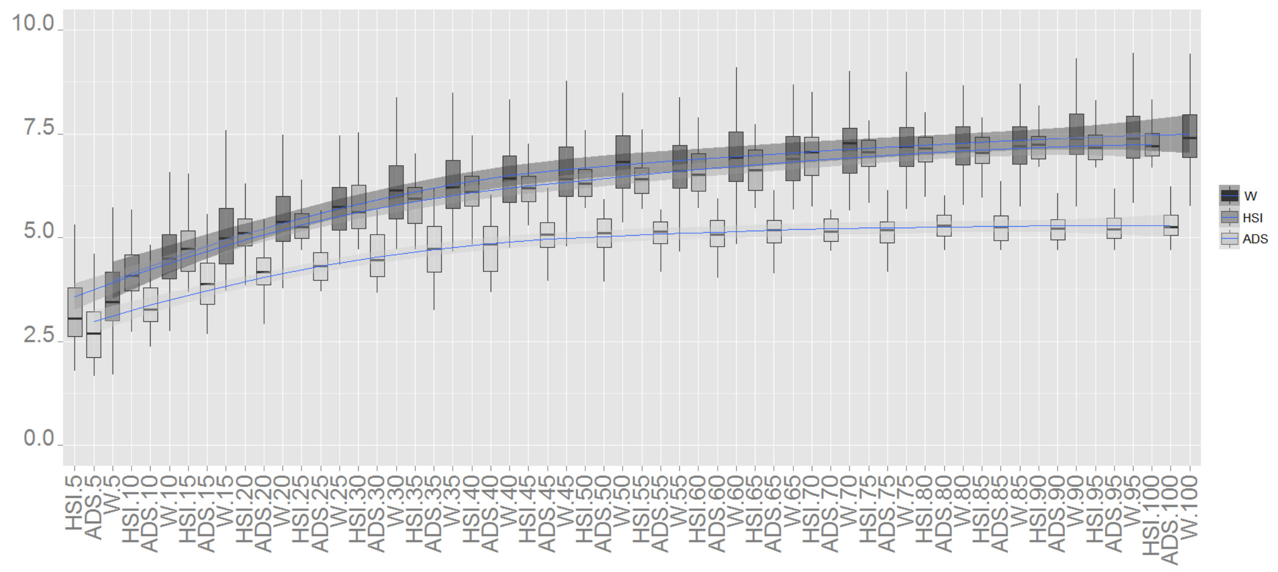

FIGURE 10: Comparison of average length of input sequences per state. Each boxplot summarises the distributions of 200 FSMs where $p = 4, q = 4$.

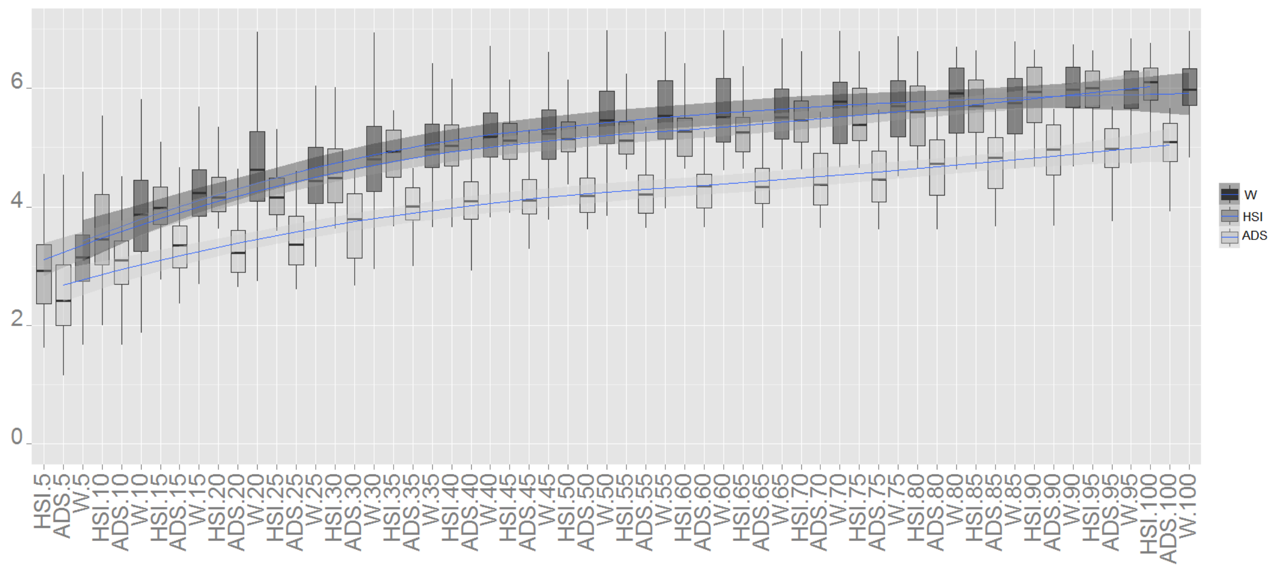

FIGURE 11: Comparison of average length of input sequences per state. Each boxplot summarises the distributions of 200 FSMs where $p = 6, q = 6$.

(incomplete) DSs that distinguish some of the states of M and such DSs might be used in automated test generation or fault localisation.

In this paper we explored the problem of constructing DSs for subsets of states of FSMs. We showed that it is PSPACE-complete to find a preset DS (PDS) that maximises the number of states distinguished and it is PSPACE-hard to approximate this problem. It is also PSPACE-complete to find a smallest set of sets of states that correspond to PDSs that distinguish all of the states of the FSM. We then explored the corresponding problems for Adaptive DSs (ADSs). It is known that we can decide in polynomial time whether an FSM has a complete ADS. However, the results for ADSs were similar to those for such PDSs: the problems considered were PSPACE-complete and it is PSPACE-hard to approximate the corresponding optimisation problem.

We then used experiments to explore the effect of optimisation by randomly generating FSMs and comparing the state identifiers produced using three methods: an algorithm for producing a characterisation set (W-generation method), a method for generating HSI sets (HSI-generation method), and ADSs returned by the greedy algorithm. The results of the experiments were promising: the greedy algorithm typically returned

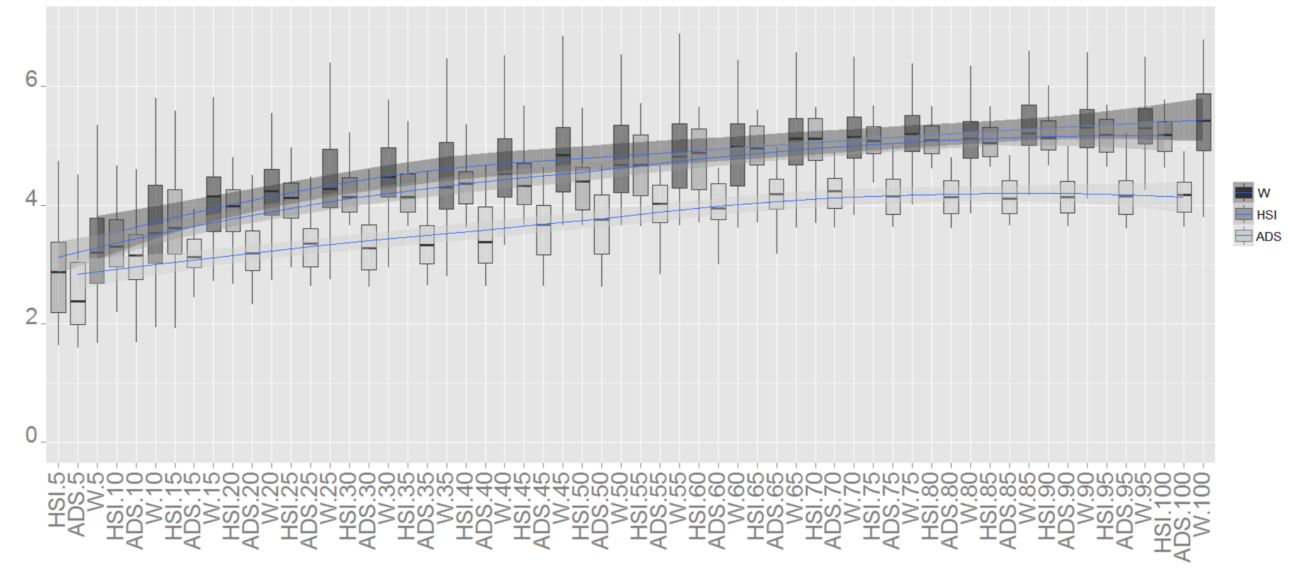

FIGURE 12: Comparison of average length of state input sequences per state. Each boxplot summarises the distributions of 200 FSMs where $p = 8, q = 8$.

Input/output (p/q) values	Corresponding $\mathcal{X}^2$–values for different number of states (n)																			
	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
4/4	25	92.57	36.15	56.57	95.29	85.16	31.4	63.33	46.93	90.2	37.06	20.57	81.56	55.63	70.67	40.04	53.1	89.15	63.91	31.89
6/6	42.06	61	28.85	52.06	98.96	35.71	29.66	85.6	56.05	43.18	38.41	47.85	53.5	42.92	20.12	33.38	97.56	96.95	79.95	22.68
8/8	77.21	92.92	60.28	70.79	89.92	81.46	51.35	41.89	63.29	93.1	65.79	92.15	72.19	99.48	94.05	61.92	66.64	25.12	21.28	57.1

TABLE 4: The results of a Kruskal-Wallis Significance Tests performed on the average number of state identifiers per state.

fewer sequences in the state identifier sets and also had smaller mean sequence length. We extended these experiments to consider 18 FSMs from a benchmark and obtained similar results. Note that a set of ADSs also defines harmonised state identifiers and we found that the results say something about the corresponding optimisation problems for harmonised state identifiers. There may well be potential to further develop this connection between the problems of generating ADSs and harmonised state identifiers.

The results have potential implications for test generation. Many techniques that generate test suites from FSMs use state identification sequences and there is the potential to use incomplete ADSs in such techniques. When we use a set H of input sequences to identify a state, the tester will typically have to separately follow an initial input sequence with each sequence from H, separating these tests with resets. Thus, typically one is interested in state identifiers that contain only a small number of input sequences. Some recently developed test generation techniques such as P [27] and H [50] are much more adaptive and it is less clear what the results in this paper tell us about such techniques.

We now outline the main contributions of this paper beyond previous work. Previous work has considered complete PDSs and ADSs, showing that existence is PSPACE-complete for PDSs and polynomial time decidable for ADSs [16]. It has also been shown that deciding whether a given set of states has an ADS is PSPACE-complete [16]. We extended the consideration of incomplete PDSs and ADSs by exploring corresponding optimisation problems. We showed that the problem of finding a maximal subset, of a given set of states, that has a PDS/ADS is PSPACE-complete. In addition, approximating these optimisation problems is PSPACE-hard. We also considered the problem of finding a smallest set of DSs that pairwise distinguish all of the states, finding that this problem is PSPACE-complete for both PDSs and ADSs. We proposed a greedy algorithm for generating a set of ADSs that pairwise distinguish the states of the specification FSM M and explored the effectiveness by generating state identifiers with the W-generation method, HSI-generation method and the incomplete ADS method.

There are several 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. There is also the question as to how effective is the greedy approach to

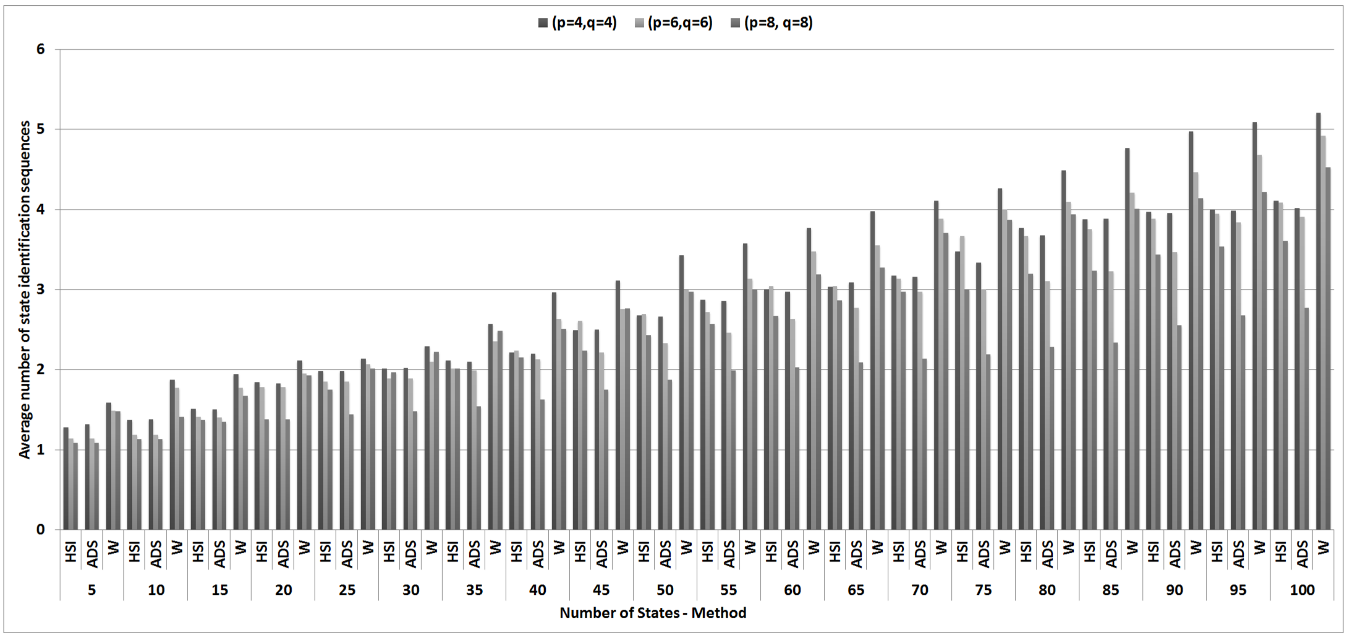

FIGURE 13: Average number of state identification sequences.

TABLE 5: Results of Case Studies.

FSM Properties					W method		HSI method		ADS method	
Name	$\|X\|$	$\|Y\|$	$\|Q\|$	$\|Q\| * \|X\|$	Length	Number of state identifiers	Length	Number of state identifiers	Length	Number of state identifiers
*dk*27	2	3	7	14	2.84	2.34	2.73	2.01	2.41	1.97
bbtas	4	4	6	24	5.31	3.25	4.95	2.99	4.55	2.46
*dk*17	4	5	8	32	5.72	2.15	5.16	1.67	5.01	1.49
*dk*15	8	3	4	32	2.34	3.64	1.98	3.41	1.40	2.43
*ex*7	11	5	5	55	1.97	4.28	1.66	3.91	1.25	3.13
mc	5	5	15	75	8.20	3.12	8.02	2.52	7.35	1.89
bbara	11	11	9	99	2.49	3.18	1.94	2.62	1.62	2.42
*dk*512	15	3	7	105	1.60	4.50	1.23	3.56	0.68	2.93
*dk*16	4	3	27	108	1.72	3.89	1.45	3.71	0.51	3.10
donfile	6	30	18	180	7.98	17.47	7.67	17.04	6.45	13.76
cse	10	5	47	470	9.63	9.44	9.23	9.30	8.52	8.50
*s*386	128	11	13	1664	7.45	7.01	6.85	6.54	6.98	6.77
bbsse	128	15	13	1664	6.34	6.01	6.02	5.32	5.61	4.91
*s*1	256	20	18	5210	9.34	4.74	8.56	4.51	7.88	3.49
planet	128	70	48	6144	14.53	3.31	13.75	3.15	11.23	2.78
*ex*1	512	18	20	10240	9.61	3.68	8.67	3.03	8.35	2.26
sand	2048	20	32	65536	17.93	13.79	17.63	13.63	15.32	12.06
nucpwr	8192	7	29	237568	10.86	18.15	10.11	17.97	8.63	15.70

generating incomplete ADSs: while the state identifiers returned were smaller than those produced using the W-generation and HSI-generation methods there may be approaches that return smaller sets of ADSs. It may be possible to improve on the greedy approach by adding a final stage in which the sets of states associated with ADSs are reduced (if these sets overlap). Finally, it would be interesting to extend this work to non-deterministic FSMs and also explore the effect of using ADSs in test generation.

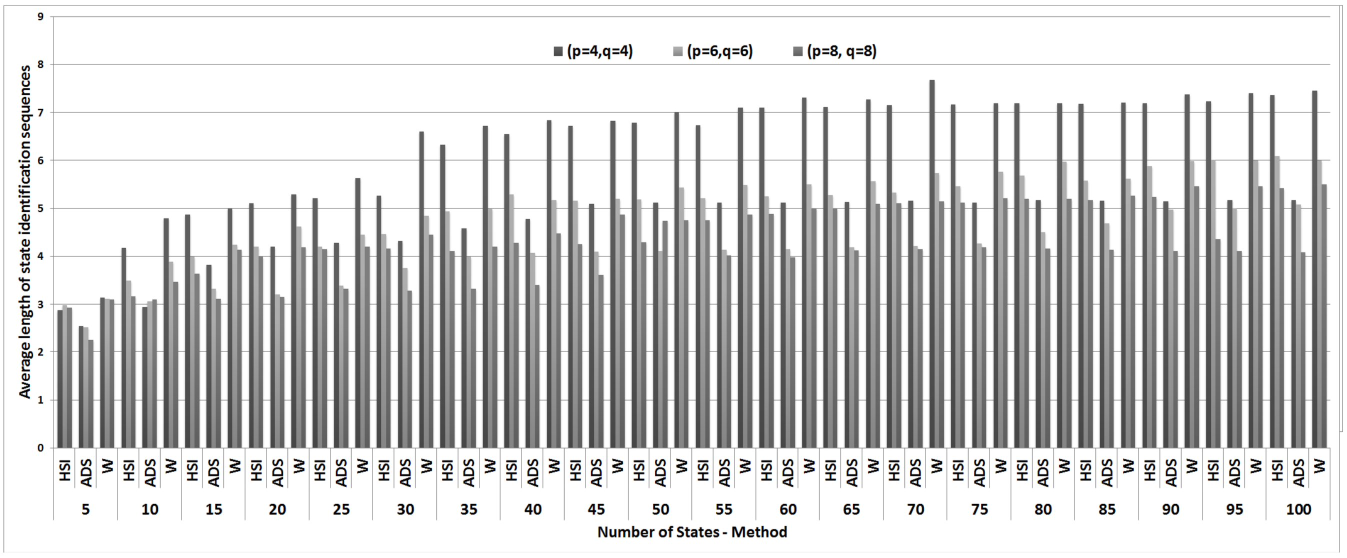

FIGURE 14: Average length of state identification sequences.

REFERENCES

[1] Friedman, A. and Menon, P. (1971) *Fault detection in digital circuits* Computer Applications in Electrical Engineering Series.

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

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

[4] Holzmann, G. J. *Design and validation of computer protocols* Prentice-Hall software series.

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

[6] Haydar, M., Petrenko, A., and Sahraoui, H. (2004) Formal verification of web applications modeled by communicating automata. *Formal Techniques for Networked and Distributed Systems (FORTE 2004)*, Madrid, September, Springer Lecture Notes in Computer Science, **3235**, 115–132. Springer-Verlag.

[7] Aho, A. V., Dahbura, A. T., Lee, D., and Uyar, M. U. (1988) An optimization technique for protocol conformance test generation based on UIO sequences and rural chinese postman tours. *Protocol Specification, Testing, and Verification VIII*, Atlantic City, New Jersey, USA, June 7-10, 75–86, Elsevier, Amsterdam, The Netherlands.

[8] Betin-Can, A. and Bultan, T. (2004) Verifiable concurrent programming using concurrency controllers. *Proceedings of the 19th IEEE international conference on Automated software engineering*, 248–257. IEEE Computer Society.

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

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

[11] Grieskamp, W., Kicillof, N., Stobie, K., and Braberman, V. A. (2011) Model-based quality assurance of protocol documentation: tools and methodology. *Software Testing, Verification and Reliability*, **21**, 55–71.

[12] Boute, R. T. (1974) Distinguishing sets for optimal state identification in checking experiments. *IEEE Trans. Comput.*, **23**, 874–877.

[13] Hierons, R. M. and Ural, H. (2006) Optimizing the length of checking sequences. *IEEE Trans. Comput.*, **55**, 618–629.

[14] Hierons, R. M., Jourdan, G.-V., Ural, H., and Yenigun, H. (2009) Checking sequence construction using adaptive and preset distinguishing sequences, . 157–166. IEEE Computer Society.

[15] Jourdan, G.-V., Ural, H., Yenigun, H., and Zhang, J. (2010) Lower bounds on lengths of checking sequences. *Formal Aspects of Computing*, **22**, 667–679.

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

[17] Hierons, R. M. and Ural, H. (2006) Optimizing the length of checking sequences. *IEEE Transactions on Computers*, **55**, 618–629.

[18] Vasilevskii, M. P. Failure diagnosis of automata. *Cybernetics and Systems Analysis*, **9**, 653–665.

[19] Luo, G., Petrenko, A., and Bochmann, G. V. (1995) Selecting test sequences for partially-specified nondeterministic finite state machines. In Mizuno, T., Higashino, T., and Shiratori, N. (eds.), *Protocol Test Systems* IFIP The International Federation for Information Processing, 95–110. Springer US.

[20] Ural, H. (1992) Formal methods for test sequence generation. *Computer Communications*, **15**, 311–325.

[21] Hierons, R. M. (1999) Minimizing the cost of fault location when testing from a finite state machine. *Computer Communications*, **22**, 120–127.

[22] da Silva Simão, A. and Petrenko, A. (2008) Generating checking sequences for partial reduced finite state machines. *TestCom/FATES*, 153–168.

[23] Yao, M., Petrenko, A., and v. Bochmann, G. (1993) Conformance testing of protocol machines without reset. *Protocol Specification, Testing and Verification, XIII (C-16)*, 241–256., Elsevier, Amsterdam, The Netherlands.

[24] Fujiwara, S. and v. Bochmann, G. (1991) Testing non-deterministic state machines with fault coverage. *Proceedings of Protocol Test Systems, IV*, 267–280.

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

[26] Hierons, R. M. and Ural, H. (2010) Generating a checking sequence with a minimum number of reset transitions. *Automated Software Engineering*, **17**, 217–250.

[27] da Silva Simão, A. and Petrenko, A. (2010) Fault coverage-driven incremental test generation. *The Computer Journal*, **53**, 1508–1522.

[28] Kohavi, Z. (1978) *Switching and Finite State Automata Theory.* McGraw-Hill, New York.

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

[30] Sokolovskii, M. N. (1971) Diagnostic experiments with automata. *Cybernetics and Systems Analysis*, **7**, 988–994.

[31] Kogan, I.V. (1973) A bound on the length of the minimal simple conditional diagnostic experiment. *Avtomatika i Telemekhanika*, **2**, 162–166.

[32] Rystsov, I.K. (1976) Proof of an achievable bound on the length of a conditional diagnostic experiment for a finite automaton. *Kibernetica*, **3**, 354–356.

[33] Natalia Kushik, N., El-Fakih, K. and Yevtushenko (2013) Adaptive homing and distinguishing experiments for nondeterministic finite state machines. *Testing Software and Systems*, Istanbul, Turkey, November 13-15, 33–48, Springer, Berlin, Heidelberg.

[34] Türker, U.C., Ünlüyurt, T. and Yenigün, H. (2014) Lookahead-based approaches for minimizing adaptive distinguishing sequences. In *Testing Software and Systems - 26th IFIP WG 6.1 International Conference, ICTSS*, Madrid, Spain, September 23-25, 32–47, Springer, Berlin, Heidelberg.

[35] Kohavi Z. and Rivierre J.A. and Kohavi, I. (1974) Checking experiments for sequential machines. *Information Sciences*, **7(0)**, 11–78.

[36] Sabnani, K., Dahbura, A. T. (1985) A new technique for generating protocol tests. *ACM Computer. Communication Review*, **15(4)**, 36–43.

[37] Chun, W. and Amer, P. D. (1992) Improvements on UIO Sequence Generation and Partial UIO Sequences. *Proceedings of the IFIP TC6/WG6.1*

Twelth International Symposium on Protocol Specification, Testing and Verification XII, Lake Buena Vista, Florida, U.S.A., 2225 June, 245–260 , Elsevier, Amsterdam, The Netherlands.

[38] Tsai, P.C. and Wang S.J. and Chang F.M. (2005) FSM-based programmable memory BIST with macro command. *Memory Technology, Design, and Testing 13th IEEE International Workshop on*, Taipei, Taiwan, August 3-5, 72–77, IEEE, New Jersey, USA.

[39] Zarrineh, K. and Upadhyaya, S.J. (1999) Programmable memory BIST and a new synthesis framework. *Fault-Tolerant Computing, Digest of Papers. Twenty-Ninth Annual International Symposium on*, Madison, Wisconsin, USA, June 15-18, 352-355, IEEE, New Jersey, USA.

[40] Xie, L. and Wei, J. and Zhu, G. (2008) An improved FSM-based method for BGP protocol conformance testing. *International Conference on Communications, Circuits and Systems*, Xiamen, China, May 25-27, 557-561, IEEE, New Jersey, USA.

[41] Drumea, A. and Popescu, C. (2004) Finite state machines and their applications in software for industrial control. *27th Int. Spring Seminar on Electronics Technology: Meeting the Challenges of Electronics Technology Progress*, Bankya, Bulgaria, May 13-16, 25-29, IEEE, New Jersey, USA.

[42] Hierons, R. M. and Türker, U. C. (2014) Distinguishing Sequences for Partially Specified FSMs. *NASA Formal Methods - 6th International Symposium, NFM 2014*, Houston, TX, USA, April 29-May 1, 62–76, Springer, Berlin, Heidelberg.

[43] Moore, E.F. (2014) Distinguishing Sequences for Partially Specified FSMs. *Automata Studies*, 129-153, Princeton University Press.

[44] Hennie, F.C. (1964) Fault detecting experiments for sequential circuits. *Proceedings of the IEEE 5th Ann. Symp. on Switching Circuits Theory and Logical Design*, Princenton, USA, November 11–13, 95-110, IEEE, New Jersey, USA.

[45] Kohavi, I. and Kohavi, Z. (1968) Variable-Length Distinguishing Sequences and Their Application to the Design of Fault-Detection Experiments. *IEEE Transactions on Computers*, **17**, 792–795.

[46] Sabnani, K. and Dahbura, A. (1988) A Protocol Test Generation Procedure. *Computer Networks*, **15**, 285–297.

[47] Gönenç, G., (1970) A Method for the Design of Fault Detection Experiments. *IEEE Transactions on Computers*, **19**, 551–558.

[48] Vuong, S. T. and Chan, W. W. L. and Ito, M. R. (1989), The uiov–method for protocol test sequence generation. *In The 2nd International Workshop on Protocol Test Systems*, Berlin , Germany.

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

[50] Dorofeeva, R. and El-Fakih, K. and Yevtushenko, N. (2005) An Improved Conformance Testing Method, *25th IFIP WG 6.1 International Conference*, Taipei, Taiwan, October 2-5,204–218.

[51] Endo, A.T. and da Silva Simão, A. (2013) Evaluating test suite characteristics, cost, and effectiveness of FSM-based testing methods. *Information & Software Technology*, **55(6)**, 1045–1062.

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

[53] Hopcroft, J. E. (1971) An n log n algorithm for minimizing the states in a finite automaton. In Kohavi, Z. (ed.), *The theory of Machines and Computation*, 189–196. Academic Press.

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

[55] Boroday, S. Y. (1998) Distinguishing tests for non-deterministic finite state machines. *Testing of Communicating Systems, IFIP TC6 11th International Workshop on Testing of Communicating Systems*, Tomsk, Russia, August 31 – September 2, 101–107. Kluwer Academic Press.

[56] Hierons, R. M. (2004) Testing from a non-deterministic finite state machine using adaptive state counting. *IEEE Transactions on Computers*, **53**, 1330–1342.

[57] Luo, G. L., v. Bochmann, G., and Petrenko, A. (1994) Test selection based on communicating nondeterministic finite-state machines using a generalized Wp-method. *IEEE Transactions on Software Engineering*, **20**, 149–161.

[58] Petrenko, A., Yevtushenko, N., Lebedev, A., and Das, A. (1994) Nondeterministic state machines in protocol conformance testing. *Proceedings of Protocol Test Systems, VI (C-19)*, Pau, France, 28-30 September, 363–378., Elsevier, Amsterdam, The Netherlands.

[59] Tripathy, P. and Naik, K. (1992) Generation of adaptive test cases from non-deterministic finite state models. *Proceedings of the 5th International Workshop on Protocol Test Systems*, Montreal, September, 309–320.

[60] Kozen, D. (1977) Lower bounds for natural proof systems. *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, Washington, DC, USA SFCS '77, 254–266. IEEE Computer Society.

[61] Savitch, W. J. (1970) Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, **4**, 177 – 192.

[62] Condon, A., Feigenbaum, J., Lund, C., and Shor, P. (1995) Probabilistically checkable debate systems and nonapproximability of PSPACE-hard functions. *Chicago Journal of Theoretical Computer Science*, **19**.

[63] Günicen, C., Türker, U. C., Ural, H., and Yenigün, H. (2012) Generating preset distinguishing sequences using sat. In Gelenbe, E., Lent, R., and Sakellari, G. (eds.), *Computer and Information Sciences II*, 487–493. Springer London. 10.1007/978-1-4471-2155-8_62.

[64] Wickham, H. (2009) *ggplot2: Elegant Graphics for Data Analysis (Use R!)*, 1st ed. 2009. corr. 3rd printing 2010 edition. Springer.

[65] Stowell, S. (2012) *Instant R: An Introduction to R for Statistical Analysis*. Jotunheim Publishing.

[66] Teetor, P. (2011) *R Cookbook*, first edition. O'Reilly.

[67] Cleveland, W. S. (1979) Robust Locally Weighted Regression and Smoothing Scatterplots. *Journal of the American Statistical Association*, **74**, 829–836.

[68] Kruskal, W. H. and Wallis, W. A. (1952) Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, **47**, 583–621.

[69] Bulmer, M. G. (1979) *Principles of Statistics*. Dover Publications.

[70] Brglez, F. ACM/SIGMOD benchmark dataset, available online at `http://cbl.ncsu.edu:16080/benchmarks/Benchmarks-upto-1996.html`.

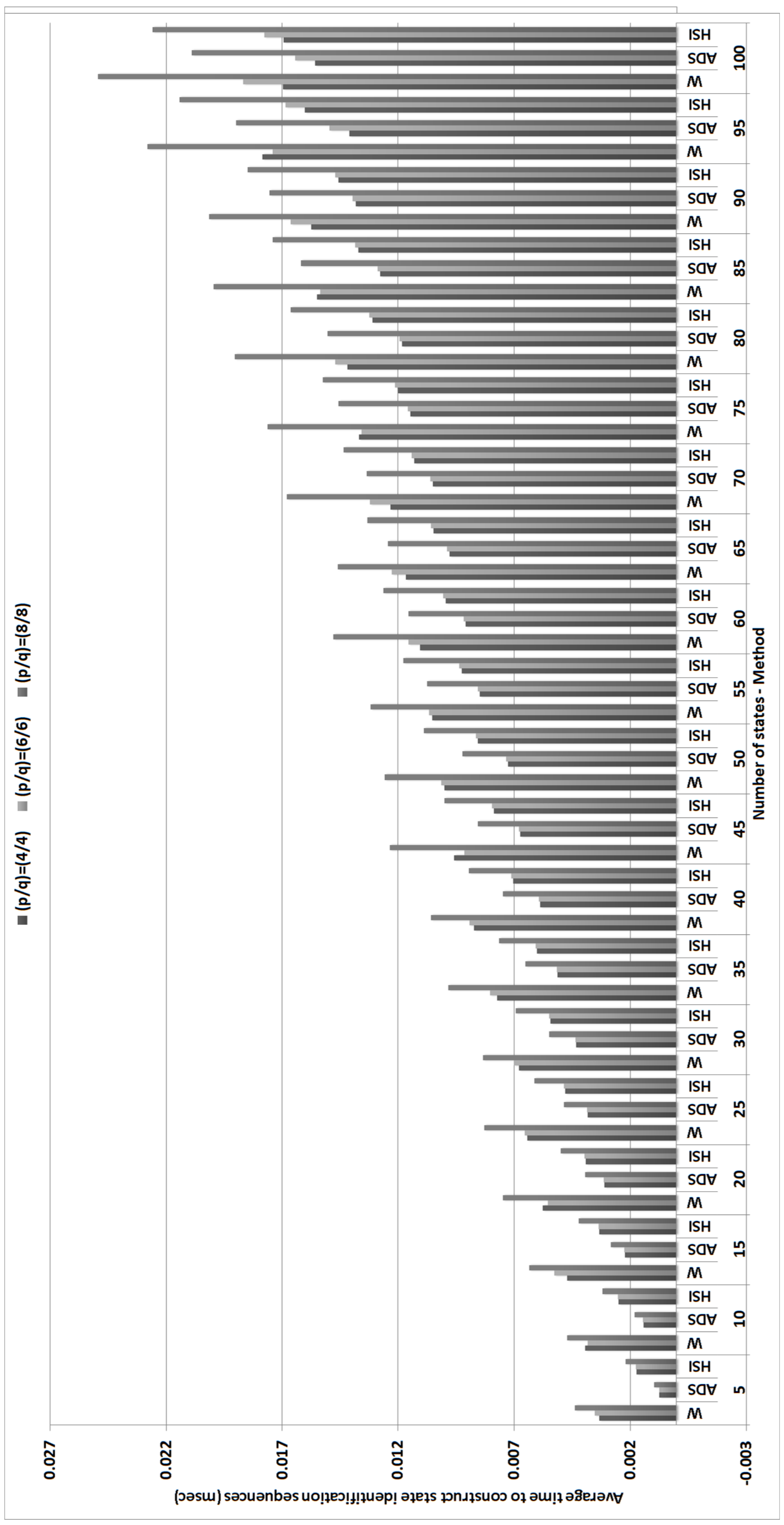

FIGURE 15: Time comparison to construct state identification sequences. Each column represents the average timings of 200 FSMs.