Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

An Experimental Study on Test Suite Characteristics, Cost, and Effectiveness of FSM-based Testing Methods

Andre Takeshi Endo
Adenilso da Silva Simao

Nº 378

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos
Junho/2012

An Experimental Study on Test Suite Characteristics, Cost, and Effectiveness of FSM-based Testing Methods

Andre Takeshi Endo, Adenilso Simao

Departamento de Sistemas de Computação (SSC)
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo (USP), Brazil
{aendo, adenilso}@icmc.usp.br

Abstract

Testing from finite state machines has been widely investigated due to its well-founded and sound theory as well as its practical application in different areas, e.g., Web-based systems and protocol testing. There has been a recurrent interest in developing methods capable of generating test suites that detect all faults in a given fault domain. However, the proposal of new methods motivates the comparison with traditional methods. In this context, we conducted a set of experiments that compares W, HSI, H, SPY, and P methods. Different parameters of the test suites were compared, such as test suite characteristics, test suite length, and fault detection ratio. These dimensions were analyzed with different machine configurations, varying number of states, inputs, outputs, and transitions. The results have shown that H, SPY, and P methods produce significantly smaller test suites than traditional methods (W, HSI). The P, H and SPY methods presented test suites of similar length, though P produced test suites which are sligthly shorter. We have also observed that the reduction ratio in partial machines is smaller than that in complete machines. On average, the recent methods produce test cases with lengths longer than traditional methods. SPY and P methods have the highest fault detection ratios and the HSI method presents the lowest ratio. Several correlations were tested and identified among the different configurations evaluated.

Keywords: Model based testing, conformance testing, experiments;

1 Introduction

The increase in the development of pervasive and mission-critical applications has demanded rigorous approaches that guarantee software reliability. Software companies have pursued solutions that have, at the same time, low cost and high effectiveness. Although formal methods and model checking have been used to formalize the software development and verification, software testing is still a necessary activity that complements these techniques by executing the system under test (SUT) and comparing the obtained behavior with the expected one. During this activity, a tester usually creates a model to guide the process of testing a system. This model is an artefact shared with other team members and useful to generate tests [1]. According to Hierons et al. [2], not only formal models or specifications can make the testing activity more effective, but also they can ease the automation. In this context, Model-Based Testing (MBT) is an approach that automatically derives test cases from a formal model built to support the testing activity. The research in MBT has reported a set of benefits, such as high fault detection ratio, reduced cost and time, traceability, and ease of handling requirements evolution [3, 4].

In MBT, the test designer can specify the test model using different modeling techniques, such as Finite State Machines (FSMs), Labeled Transition Systems, and UML state machines. FSM is a formal modeling technique adopted due to its rigor and simplicity, and used mainly in the testing of protocols and reactive systems [5, 6]. Although FSM-based testing has been studied for several decades [7, 8], recent contributions in the area have still been achieved [9, 10, 11, 12, 13]. Great effort has been spent on the development of methods that generate effective test suites, i.e., methods that reveal as many faults as possible. A so-called complete test suite, capable of revealing all faults from a given fault domain in an implementation, can be generated from a specification if some assumptions are made. One of these assumptions is that the maximum number of states in the implementation is known. When both specification and implementation have the same number of states, the generated test suite is called n-complete [14]. There are several methods in the literature that generate n-complete test suites, such as W [15, 16], HSI [17], H [14], SPY [18], and P [9]. These methods differ mainly with respect to the length of generated test suites (cost) and the complexity of the adopted algorithms. As the effectiveness is theoretically guaranteed by all methods, cost is the main measure for comparison. As a consequence, researchers have focused on generating shorter test suites.

Traditional methods, such as W and HSI, are characterized by the concatenation of transition cover sets with statically created identification sets. The effectiveness of these methods is demonstrated by the test suite construction. Another research line is the identification of sufficient conditions that determine whether a test suite is n-complete or not [19, 14, 10, 9]. Recent methods, like H, SPY and P rely on sufficient conditions to support the test case generation. The H method [14] was the first method to use sufficient conditions explicitly. Thus, shorter test suites can be generated compared with previous methods since certain properties are tested *on-the-fly* and more adequate sequences may be selected. The SPY method, based on sufficient conditions weaker than those of H, analyzes the test suite that is being generated and chooses sequences aiming to reduce the number of sequences in the resulting test suite [18]. The method presents good results if the implementation has more states than the specification. Recently, the P method was developed to support incremental testing; it relies on the weakest sufficient conditions currently available in the literature[9]. Recent methods are difficult to compare since they rely on heuristics that produce different results according to the FSM characteristics. Therefore, experimental evaluations have been conducted to identify trade-offs among methods that may not be theoretically compared [11]. Although the papers that have proposed new methods present some experimental results (e.g., [14, 18, 9]), there are a few studies on the comparison among different methods [11]. Dorofeeva et al. [11] experimented with a set of well-known methods, evaluating different aspects, such as test suite length, derivation time, and complexity. However, the first version of this study was published in 2005 [20] and, as a consequence, recent methods (SPY, P) were not compared. Moreover, only completely specified FSMs were considered and different variations, such as number of inputs, outputs, and states in FSMs were not analyzed individually.

We compare the test suites generated automatically using the methods W, HSI, SPY, H, and P. First, for test suites derived using the compared methods, we analyzed the number of test cases and their length. Second, the total cost (i.e., the length) of each test suite was compared. Third, the effectiveness of the methods was analyzed using the mutant analysis in a scenario in which the implementation's number of states was incorrectly estimated. Finally, we discuss correlations between the different dimensions analyzed.

This technical report shows an experimental study that compares traditional and recent FSM generation methods that have not been compared yet. Its main contribution is an experimental evaluation of the main methods found in the literature, considering the W, HSI, H, SPY, and P methods. The results have shown that the P method is able to generate test suites with shorter length compared with other methods. With complete FSMs, the difference is greater in comparison to traditional methods and smaller concerning recent methods. With partial FSMs, the reduction ratio among the methods is smaller, which indicates that results with complete specifications cannot be generalized to partial ones.

Previous results of this research were published in [21]. The experimental evaluation was extended by (i) analyzing the characteristics of test cases found in the different test suites; (ii) evaluating the fault-detection ratio using mutant analysis; and (iii) correlating these different dimensions for each applied method.

The remainder of this paper is organized as follows: Section 2 presents definitions of FSMs and adopted notations; Section 3 describes the methods compared in the experiments, as well as an example to illustrate the test suite construction; Section 4 shows experiments setting; Section 5 shows the results w.r.t. experiments we conducted; Section 6 presents a discussion about the obtained results; Section 7 shows the related work; finally, Section 8 presents the conclusion and discusses future work.

2 Preliminaries

A Finite State Machine (FSM) is a deterministic (Mealy) machine, defined as follows [10]. An FSM M is a 7-tuple$(S, s_0, I, O, D, \delta, \lambda)$, where:

- S is a finite set of states with initial state s_0,
- I is a finite set of inputs,
- O is a finite set of outputs,
- $D \subseteq S \times I$ is a specification domain,
- $\delta : D \to S$ is a transition function, and
- $\lambda : D \to O$ is an output function.

The tuple $(s, x) \in D$ is a *defined transition* in state s that consumes input symbol x. The form (s_i, x, y, s_j) can also be used to represent a transition from s_i to s_j that consumes input x and produces output y. In a transition $t = (s_i, x, y, s_j)$, s_i is said to be the head state and s_j the tail state; t is an incoming transition for s_j and t is an outgoing transition for s_i. An FSM which has defined transitions for each input symbol in all states, i.e., $D = S \times I$, is complete. Otherwise, the FSM is partial. A sequence $\alpha = x_1...x_k \in I^*$ is an input sequence defined for the state $s \in S$, if there exist $s_1, ..., s_{k+1}$ such that $s = s_1$ and $\delta(s_i, x_i) = s_{i+1}$ for all $1 \leq i \leq k$; we say that α is a *transfer sequence* from s to s_{k+1} and that s_{k+1} is *reachable* from s. An FSM is strongly connected if every state is reachable from all states. An FSM is initially connected if every state is reachable from the initial state s_0. Figure 1 shows an example of a state diagram representation of an FSM.

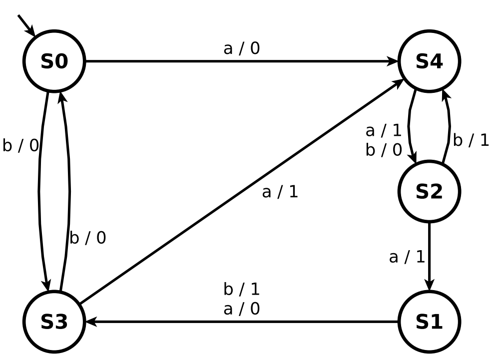


Fig. 1: Example of an FSM.

Notation $\Omega(s)$ is used to denote all input sequences defined for state s and Ω_M as an abbreviation for $\Omega(s_0)$. Therefore, Ω_M represents all defined sequences for the FSM M. In this work, we assume that the FSM has a reset operation that brings the machine to its initial state. The reset operation is denoted by r. Notation $\alpha\omega$ represents the concatenation of the two sequences, α and ω. Sequence α is prefix of sequence β, denoted by $\alpha \leq \beta$, if $\beta = \alpha\omega$, for some sequence ω. Sequence α is proper prefix of β, denoted by $\alpha < \beta$, if $\beta = \alpha\omega$ for some $\omega \neq \epsilon$. Given two sets of sequences D_1 and D_2, $D_1.D_2$ is the set of sequences obtained by concatenating all sequences in D_1 with all sequences in D_2, i.e., $D_1.D_2 = \{\alpha\beta \mid \alpha \in D_1 \text{ and } \beta \in D_2\}$. Given a set T, the notation pref(T) represents all the prefixes of sequences in T, i.e., pref$(T) = \{\alpha \mid \beta \in T \text{ and } \alpha \leq \beta\}$. If $T = $ pref(T), then we say that T is prefix-closed.

The transition and output functions are extended for defined input sequences, including the empty sequence ϵ, as follows. For a state $s_i \in S$, $\delta(s_i, \epsilon) = s_i$ and $\lambda(s_i, \epsilon) = \epsilon$, given an input sequence $\alpha x \in \Omega(s_i)$, we have $\delta(s_i, \alpha x) = \delta(\delta(s_i, \alpha), x)$ and $\lambda(s_i, \alpha x) = \lambda(s_i, \alpha)\lambda(\delta(s_i, \alpha), x)$.

Two states $s_i, s_j \in S$ are *distinguishable* if there exists a *separating sequence* $\gamma \in \Omega(s_i) \cap \Omega(s_j)$, such that $\lambda(s_i, \gamma) \neq \lambda(s_j, \gamma)$. An FSM M is reduced (or minimal) if all states are pairwisely distinguishable. Given a different FSM $N = (Q, q_0, I, O, \Delta, \Lambda)$, we say that two machines M and N are distinguishable if there exists a sequence $\gamma \in \Omega_M \cap \Omega_N$, such that $\lambda(s_0, \gamma) \neq \Lambda(q_0, \gamma)$.

A test case of M is an input sequence $\alpha \in \Omega_M$ starting with symbol r. A test suite of M is a finite set of test cases of M, such that there are no two test cases α and β, such that $\alpha < \beta$. In fact, if a test case α is a proper prefix of a test case β, the execution of β will always imply the execution of α, and can thus be removed without altering the test result. The number of symbols (length) of a sequence α is represented by $|\alpha|$. This notation is extended to represent the length of all sequences in a test suite T, $|T|$.

Assuming a specification M, we denote by $\Im$ the set of all deterministic FSMs with the same input alphabet as M for which all sequences in Ω_M are defined, i.e., for each $N \in \Im$, $\Omega_M \subseteq \Omega_N$. Set $\Im$ is called a *fault domain* for M. Let $m \geq 1$ be an integer, $\Im_m$ denotes all FSMs of $\Im$ with at most m states. Given a specification M with n states, a test suite $T \subseteq \Omega_M$ is m-complete, $m \geq n$, if, for each $N \in \Im_m$ distinguishable from M, there exists a test in T that distinguishes M from N. An m-complete test suite has *full fault coverage* for the defined domain and is able to detect all faults in any implementation with at most m states. In this work, we consider n-complete test suites that represent the case in which $m = n$.

A set of input sequences Q is a *state cover* of M if, for each state $s_i \in S$, there exists a sequence $\alpha_i \in Q$ that transfers the FSM from the initial state to s_i. This set includes sequence ϵ to reach the initial state. A set of input sequences P is a *transition cover* of M if, P includes the empty sequence ϵ and for each transition $(s, x) \in D$ there exist the sequences $\alpha, \alpha x \in P$ such that $\delta(s_0, \alpha) = s$.

Sets Q and P can be obtained by means of paths in the testing tree of M. Figure 2 shows the testing tree for the FSM in Figure 1. Basically, the tree is built by expanding each state once, starting in the initial state. A procedure to build a testing tree of an FSM can be found in [16].

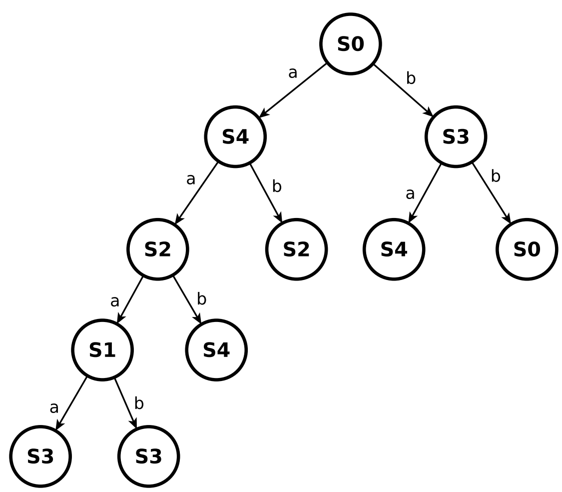


Fig. 2: Testing tree for calculating Q and P.

Using a breadth-first search in this tree that visits all states once, the state cover set $Q = \{\epsilon, a, b, aa, aaa\}$ is produced. Using a depth-first or breadth-first search that visits all nodes in this tree, the transition cover set $P = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aaaa, aaab\}$ is produced.

To identify states and check transitions, traditional methods use pre-defined sets, such as characterization sets and separating families. A *characterization set*, also known as W set, is a set of defined input sequences that contains at least a separating sequence for each pair of states in the FSM. A *separating family* is a set of state identifiers H_i for a state $s_i \in S$ that satisfies the following condition [17]: for any two different states s_i, s_j, there exist sequences $\beta \in H_i$ and $\gamma \in H_j$ that have a common prefix α, i.e., $\alpha \leq \beta$ and $\alpha \leq \gamma$, such that $\alpha \in \Omega(s_i) \cap \Omega(s_j)$ and $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$. Polynomial algorithms to identify characterization sets and separating families can be found in [8, 17].

Recent methods, such as H, SPY, and P rely on sufficient conditions to support test case generation. These sufficient conditions provide mechanisms to check if a given test suite is m-complete. However, a test suite that does not satisfy these conditions may be m-complete, i.e., these conditions are not *necessary*. The sufficient conditions for the test suites found in the literature are described in [14, 10, 9] and used by the H, SPY, and P methods, respectively.

2.1 Mutant Analysis for FSMs

To model faults in the implementation, we use mutants of the specification. Given an FSM $M = (S, s_0, I, O, D, \delta, \lambda)$, the operators generate a mutant N and are defined as follows:

- Operator *Change Initial State* (op_{CIS}) changes initial state s_0 to s_k, such that $s_k \in S$ and $s_k \neq s_0$. Thus, the mutant $N = (S, s_k, I, O, D, \delta, \lambda)$ is created. Formally, $op_{\text{CIS}}(M) = \{N = (S, s_k, I, O, D, \delta, \lambda) \mid s_k \in S \setminus \{s_0\}\}$.

- Operator *Change Output* (op_{CO}) modifies the output of a transition (s, x), creating a different output function Λ. Thus, a mutant $N = (S, s_0, I, O, D,\ \delta, \Lambda)$ is created, such that for the transition $(s, x) \in D$, $\lambda(s, x) \neq \Lambda(s, x)$. Formally, $op_{\text{CO}}(M) = \{N = (S, s_0, I, O, D, \delta, \Lambda) \mid \exists (s, x) \in D,\ y \in O \setminus \{\lambda(s, x)\},\ \forall (s', x') \in D \setminus \{(s, x)\},\ \Lambda(s', x') = \lambda(s', x'),\ \Lambda(s, x) = y\}$.

- Operator *Change Tail State* (op_{CTS}) modifies the tail state of a transition (s, x), creating a different transition function Δ. Thus, a mutant $N = (S, s_0, I, O, D, \Delta, \lambda)$ is created, such that for a transition $(s, x) \in D$, $\delta(s, x) \neq \Delta(s, x)$. Formally, $op_{\text{CTS}}(M) = \{N = (S, s_0, I, O, D, \Delta, \lambda) \mid \exists (s, x) \in D,\ s_k \in S \setminus \{\delta(s, x)\},\ \forall (s', x') \in D \setminus \{(s, x)\},\ \Delta(s', x') = \delta(s', x'),\ \Delta(s, x) = s_k\}$.

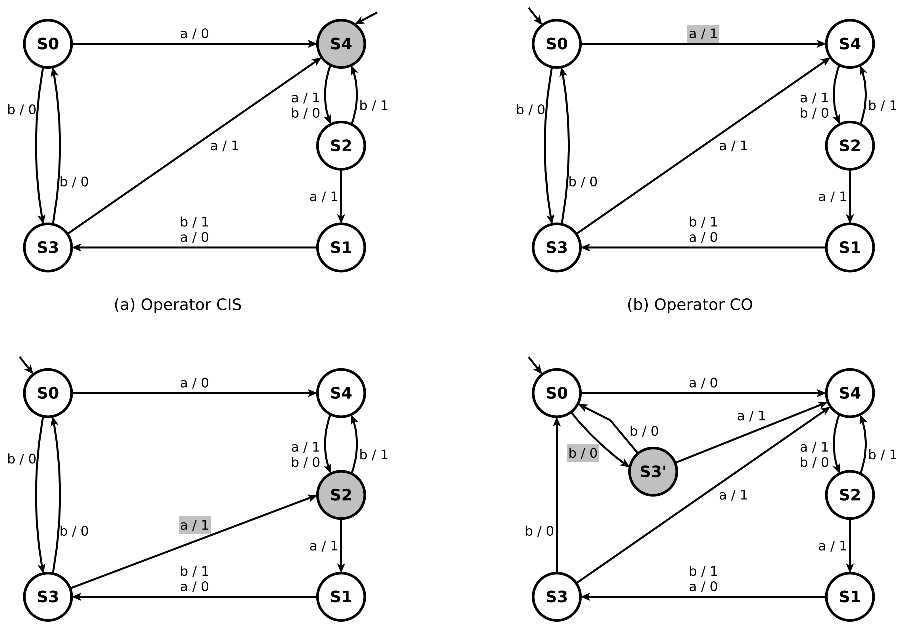
- Operator *Add Extra State* (op_{AES}) inserts a new state so that mutant N is equivalent to M. To generate a mutant $N = (S_N, s_0, I, O, D_N, \Delta, \Lambda)$, such that for a state $s \in S$, s has at least two incoming transitions, create a new state s_n and copy its outgoing transitions, redirect one transition from s to s_n. For any input $x \in I$ defined in s, $\Delta(s\prime_n, x) = \delta(s, x)$ and $\Lambda(s_n, x) = \lambda(s, x)$; $S_N = S \cup \{s_n\}$. Formally, $op_{\text{AES}}(M) = \{N = (S_N, s_0, I, O, D_N, \Delta, \Lambda) \mid \exists s, s_i, s_j \in S, x_i, x_j \in I, x_i \neq x_j,\ \delta(s_i, x_i) = \delta(s_j, x_j) = s,\ S_N = S \cup \{s_n\},\ D_N = D \cup \{(s_n, x) \mid (s, x) \in D\},\ \Delta(s_i, x_i) = s_n,\ \Lambda(s_i, x_i) = \lambda(s_i, x_i),\ \forall (s', x') \in D \setminus \{(s_i, x_i)\},\ \Delta(s', x') = \delta(s', x'),\ \Lambda(s', x') = \lambda(s', x'),\ \forall (s, x) \in D,\ \Delta(s_n, x) = \delta(s, x),\ \Lambda(s_n, x) = \lambda(s, x)\}$.

Figure 3 shows examples of mutants for the FSM presented in Figure 1, highlighting the modifications in grey. Initial state s_0 is changed by s_4 in Figure 3(a) using operator op_{CIS}. In Figure 3(b), operator op_{CO} modifies the output for transition (s_0, a). In Figure 3(c), the tail state for transition (s_3, a) is redirected from s_4 to s_2 using operator op_{CTS}. In Figure 3(d), operator op_{AES} adds extra state s'_3 based on s_3 and its outgoing transitions. Transition (s_0, b) is redirected from s_3 to s'_3.

Those examples are called first-order mutants since the operators are applied just once on the original FSM. When operators are applied two or more times in the original program (specification), the resulting mutants are of higher order [22]. Given an FSM M, the application of one operator op_1 produces a first order mutant N_1, i.e., $N_1 \in op_1(M)$. By applying a mutant operator on N_1, a second-order mutant is generated and so on. Figure 4 shows a second-order mutant. Operator op_{AES} adds an extra state s'_3 and operator op_{CO} modifies the output for transition (s_2, a).

A classical problem in mutant analysis is the detection of equivalent mutants. A mutant is equivalent to the original if it always produce the same output as the original. In FSMs, this problem can be solved in polynomial time. It can be cast as the problem of finding a separation sequence between the initial states of the two known FSMs. If this sequence does not exist, the two machines are equivalent. A procedure to find a separation sequence between two FSMs is presented in Algorithm 1.

As the number of states of M and N are finite, the possible state pairs (s_i, q_i) are also finite. As the number of pairs is finite and a pair is not visited twice (since they are saved in V), the algorithm always stops. If γ exists, the algorithm will identify it by incrementally checking and increasing the



(a) Operator CIS (b) Operator CO

(c) Operator CTS (d) Operator AES

Fig. 3: Examples for mutant operators.

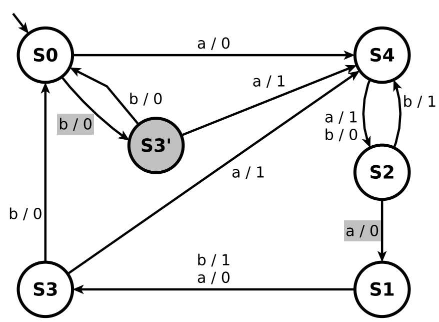


Fig. 4: 2-order mutant.

test sequence length. An already visited pair does not need to be verified since the sequences up to there were also checked. This algorithm is inspired by the state spliting into equivalent blocks [5] and the distinguishing machine [23].

Algorithm 1 Find separation sequence between two FSMs.

```
input: M = (S, s_0, I, O, D, δ, λ), and N = (Q, q_0, I, O, Δ, Λ)
output: separation sequence γ if it exists; ϵ, otherwise.

initialize a queue q
initialize a set of state pairs V = {}
q.enqueue(ϵ)
while q.isNotEmpty() do
  γ =q.dequeue()
  if λ(s_0, γ) ≠ Λ(q_0, γ) then
    return γ
  else
    s_i = δ(s_0, γ)
    q_i = Δ(q_0, γ)
    if (s_i, q_i) ∉ V then
      V = V ∪ {(s_i, q_i)}
      foreach x ∈ I do
        q.enqueue(γx)
      endfor
    endif
  endif
endwhile

return ϵ
```

3 Test Generation Methods

In this section, we briefly introduce the W, HSI, H, SPY, and P methods used in the experiments. For each method, an overview of the test suite construction for the example in Figure 1 is also provided. All these methods are applicable to initially connected, reduced, complete, and deterministic machines. HSI, H and P methods are also applicable to partial FSMs.

3.1 W

The works of Vasilevskii's [15] and Chow's [16] are the seminal papers in test case generation methods for FSMs, proposing the W method. The method uses the transition cover set P to reach states and transitions and the characterization set W for state identification. If $m = n$, a test suite is generated by the concatenation of sets P and W and removal of the proper prefixes.

For the example in Figure 1, given that $P = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aa\ aa, aaab\}$ and $W = \{a, bb\}$, after $P.W$ and removing the sequences which are prefixes of other tests, we obtained the test suite $\mathrm{TS_W} = \{raba, rabbb, rbaa, rbabb, rbba, rbbbb, raaba, raabbb, raaaaa, raaaabb, raaaba, raaabbb\}$, $|\mathrm{TS_W}| = 64$.

3.2 HSI

The HSI method [17] uses the separating family to check states in both state identification and transition testing. The separating family can be obtained from a characterization set W, which, in the worst case, will be the W set itself.

In HSI, given the same P and the harmonized state identifiers $H_0 = \{a, b\}$, $H_1 = \{a, b\}$, $H_2 = \{a, b\}$, $H_3 = \{ab\}$, $H_4 = \{ab\}$, the test suite is obtained by concatenating all sequences $\alpha \in P$ with H_i such that $\delta(s_0, \alpha) = s_i$. After removing proper prefixes, we obtained the test suite $\mathrm{TS_{HSI}} = \{raba, rabb, rbab, rbba, rbbb, raabb, rbaab, raaaab, raaabb, raabab, raaaaab, raaabab\}$, $|\mathrm{TS_{HSI}}| = 62$.

3.3 H

Similarly to the HSI method, the H method [14] also adopts separating families and can be seen as an improved method. The difference is that the H method selects state identifiers *on-the-fly* during the transition testing phase [14]. Using this strategy, the method is able to reduce the length of the test suites produced.

The first part of the method adds sequences that distinguish reached states for each pair of sequence in Q. Given the state cover $Q = \{\epsilon, a, b, aa, aaa\}$ and the separating family $H_0 = \{a, b\}$, $H_1 = \{a, b\}$, $H_2 = \{a, b\}$, $H_3 = \{a, ba\}$, $H_4 = \{a, ba\}$, the first part is equivalent to $Q.H_i = \{ba, bba, aba, aab, aaaa, aaab\}$, after removing prefixes. Notice that the separating family used here is different from the one used in the HSI method. In the second part, transitions are verified w.r.t. the sufficient conditions. If transitions are not verified, adequate state identifiers are selected *on-the-fly*. For instance, to check transition (s_3, a) we have sequence ba such that $\delta(s_0, ba) = s_4$. We can apply the previously defined state identifier H_4, but sequence aaa has been chosen for this task. The resulting test suite for H method is $\mathrm{TS_H} = \{rbba, rbbb, rabaa, rbaaaa, raaabaaa, raabaaa, raaaaaaa\}$, $|\mathrm{TS_H}| = 42$.

3.4 SPY

The SPY method [18] was proposed to reduce the length of the test cases, using a strategy that shortens the number of test cases by avoiding creating new branches in the test suite. Thus, the SPY method also chooses state identifiers *on-the-fly* so that it avoids creating new branches by the extension of existing test cases. Using SPY, we obtained the test suites $\mathrm{TS_{SPY}} = \{raaaaab, raaaab, raaabab, raaabb, raabab, raba, rbabb, rbba, rbbbaabb\}$, $|\mathrm{TS_{SPY}}| = 53$.

3.5 P

The P method [9] was initially proposed to obtain an n-complete test suite by incrementing a user-defined test suite provided. However, if we consider that the user-defined test suite contains only the empty sequence, the P method can be used to generate n-complete test suites in the traditional way. The P method is also able to choose sequences *on-the-fly* to distinguish states. The method iteratively checks sufficient conditions and applies defined rules to derive the test suite.

The P method basically works in two steps. First, it builds a test set so that the n states are distinguished. In the second step, the method checks the transitions not confirmed by the sufficient conditions. For each new test added to the test set, rules can be triggered and other transitions can be checked. At the end, after removing prefixes and adding resets, we have the final test suite $\mathrm{TS_P} = \{rababba, raabba, rabaaab, rabba, rbabb, rbbb\}$, $|\mathrm{TS_P}| = 34$.

4 Experimental Study

We conducted experiments to evaluate test suites generated by different methods, from traditional (W, HSI) to recent ones (H, SPY, and P). We aimed at answering the following research questions:

- RQ1: What are the characteristics of the test suite generated by different methods, concerning number of resets, average test case length, maximum and minimum lengths?
    1. How does the number of states impact on these characteristics?
    2. How does the number of inputs impact on these characteristics?
    3. How does the number of outputs impact on these characteristics?
    4. How does the completeness degree impact on these characteristics?
- RQ2: Which method has the best cost, i.e., test suite length? Are their cost differences significant?
    1. How does the test suite length behave when the number of states varies?
    2. How does the test suite length behave when the number of inputs varies?

3. How does the test suite length behave when the number of outputs varies?
4. How does the test suite length behave when the completeness degree varies?

- RQ3: Which method is more effective (higher fault detection) when the implementation has more states than the specification?

1. How does the fault detection ratio behave when the number of states varies?
2. How does the fault detection ratio behave when the number of inputs varies?
3. How does the fault detection ratio behave when the number of outputs varies?
4. How does the fault detection ratio behave when the completeness degree varies?

- RQ4: How are test case characteristics, test suite length and fault detection ratio correlated?

Reduced and deterministic FSMs were randomly generated, varying the number of states, inputs, outputs, and transitions. For each FSM configuration (#states, #inputs, #outputs, and #transitions), 100 machines were generated and the average measures were calculated. In total, 5200 FSMs were used in this experiment. The raw data collected and analyzed in this paper are available on the Web (see `http://www.labes.icmc.usp.br/~aendo/fsm-experiments`) [1]. For each test suite, the shortest and longest test case were identified, as well as the average test case length. For a given configuration, the average for these data was also calculated.

We compared the cost of the methods w.r.t. the test suite length (with resets and no proper prefixes). The reduction and reduction ratio were calculated comparing the test suite length with the W method. Given a method μ, the reduction ratio was calculated using the W method as a reference, i.e., $|\mathrm{TS}_\mu|/|\mathrm{TS}_\mathrm{W}|$, where$|\mathrm{TS}_\mu|$ is the average test suite length produced by method μ.

Regarding partial FSMs, we compared the traditional HSI method with the recent P method. From the traditional methods evaluated in this paper, only HSI is applicable to partial FSMs. The P method was selected as it showed the best performance among the recent methods regarding the test suite length. We also use the notion of *completeness degree* which is calculated dividing the number of transitions (in the partial FSM) by the number of transitions in a complete FSM with same number of states and inputs.

All compared methods are able to generate n-complete test suites. To evaluate the fault detection, we consider a scenario such that n-complete test suites will not detect all faults, i.e., when the implementation is not on the fault domain $\Im_n$. We use a strategy based on high-order mutation testing. First, a mutant m_1 with $n+1$ states was generated using operator op_AES. Second, other mutants were produced applying operators op_CIS, op_CO, and op_CTS to m_1. Thus, second order mutants were used to evaluate the fault detection of test suites generated by the methods. We used Algorithm 1 to eliminate the equivalent mutants.

In this study, four types of configuration were used:

1. Number of states: we generated complete FSMs with four inputs, four outputs, and number of states ranging from four to 30.

2. Number of inputs: we generated complete FSMs with ten states, four outputs, and number of inputs ranging from two to ten.

3. Number of outputs: we generated complete FSMs with ten states, four inputs, and number of outputs ranging from two to ten.

4. Completeness degree: we generated partial FSMs with ten states, four inputs, four outputs, and number of transitions ranging from 20 (0.50) to 39 (0.98). The numbers between parentheses is the completeness degree.

[1] All FSMs and test suites produced for each method are available. Future comparisons can be supported by these resources.

The statistical significance between two methods was calculated using Wilcoxon test. If no information is provided, you can assume that the obtained p-value is $< 2.2e^{-16}$ (R statistical tool). Moreover, we also provide measures to analyze the effect size. Effect size measures are used to complement hypothesis tests, indicating practical significance, importance, or meaningfulness [24]. To calculate the effect size, we used (i) unstandardized measures represented by median and mean differences; and standardized measures represented by Standard Mean Difference (SMD) d and Hedges' g. When comparing two variables for each method, three correlation coefficients were used: *Pearson Correlation Coeficient* (PCC), *Spearman Correlation Coeficient* (SCC), and *Kendall Correlation Coeficient* (KCC). Correlation coefficients are used to measure the dependence/relationship between two variables. Their values ranges from -1 to $+1$, being $+1$ a perfect positive correlation and -1 a perfect negative correlation. The closer the coefficient is to either -1 or $+1$, the stronger the correlation between the variables. Positive correlation means that as one variable increases, the other variable tends to increase; and in the negative correlation, if one variable increases, the other variable tends to decrease. Values close to zero suggest that the variables are not linearly correlated (by using Pearson). Spearman's and Kendall's coefficients are able to detect correlations without requiring a linear relationship.

We followed guidelines for statistical analysis of software engineering experiments proposed by [25, 24]. We used the toolkits R [26] and Gnuplot [27] to support the statistical analysis provided in this work. In R, the libraries "compute.es" and "MBESS" were used to calculate standard mean difference measures d and Hedges' g.

5 Analysis of Results

5.1 RQ1: Test Suite Characteristics

In this section, we analyzed four metrics concerning the test suites generated for each method. Given a method μ, we define:

- Number of resets (NR_μ) as the number of resets in a test suite. It also represents the number of test cases since there exists a reset before each test case.
- Minimum test case length (minL_μ) as the length of the shortest test case w.r.t. the number of executed inputs.
- Maximum test case length (maxL_μ) as the length of the longest test case w.r.t. the number of executed inputs.
- Average test case length (avgL_μ) as the average length of test cases in a test suite.

5.1.1 Number of Resets

Table 1 shows the correlation between number of resets and states, inputs, outputs, and transitions for each method. The results are discussed along with the following graphs.

Tab. 1: Correlation between number of resets and states, inputs, outputs, and transitions.

#Resets	States			Inputs			Outputs			Transitions		
	PCC	SCC	KCC	PCC	SCC	KCC	PCC	SCC	KCC	PCC	SCC	KCC
W	.97	.98	.92	.92	.93	.83	-.74	-.74	-.64	-	-	-
HSI	.99	.99	.94	.98	.98	.91	-.84	-.85	-.71	.91	.92	.77
SPY	.98	.98	.92	.96	.96	.87	-.73	-.73	-.57	-	-	-
H	.98	.98	.91	.96	.96	.87	-.73	-.71	-.56	-	-	-
P	.98	.98	.92	.96	.97	.89	-65	-.63	-.49	.83	.81	.65

Figure 5 shows how the number of resets varies in function of the number of states. All methods present an approximated linear growth, having W the highest number of resets and P the lowest.

The SPY and H methods have a quite similar number of resets, becoming different only after 20 states. For the three coefficients shown in Table 1, there exists a strong positive correlation between the number of resets and number of states (over 0.91). All methods presented similar correlation coefficients.

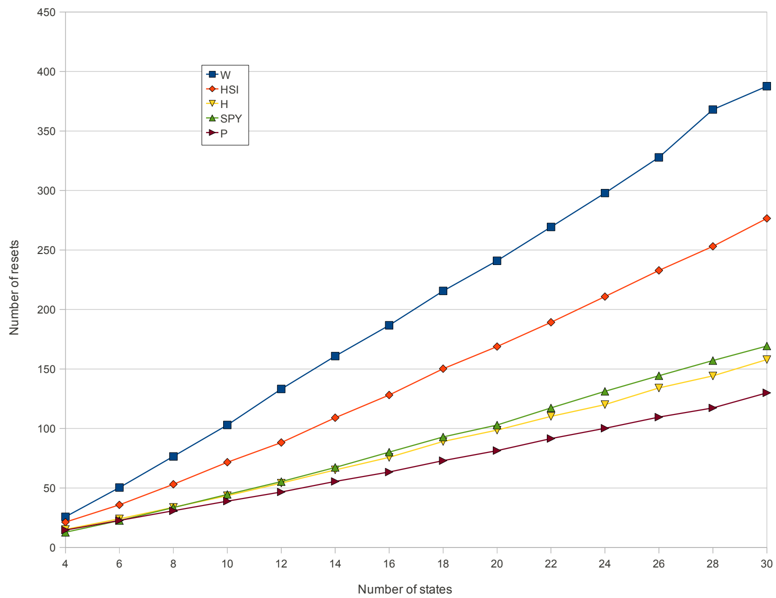


Fig. 5: Number of resets varying number of states.

Figure 6 shows how the number of resets varies in function of the number of inputs. The behavior of each method is similar to the presented one in Figure 5, with the following order $NR_W > NR_{HSI} > NR_H > NR_{SPY} > NR_P$. Notice that, in this configuration, SPY has fewer resets than H, being very close to the P method. A strong positive correlation is also observed between number of resets and inputs (over 0.83 for the three coefficients in Table 1). The coefficients of W is slightly smaller than other methods.

Figure 7 shows how the number of resets varies in function of the number of outputs. There is a decrease in the number of resets, which is smaller and close for the SPY, H and P methods. The traditional methods show a greater decrease in the number of resets. There is a negative correlation between number of resets and number of outputs (under -0.49 for the three coefficients in Table 1). However, there is more variation among the methods and the correlation is generally weaker than in previous analysis (states, inputs).

Figure 8 shows how the number of resets varies in function of the completeness degree (number of transitions). The HSI method has more resets and presents a constant growth. On the other hand, the P method has a growth until 0.85 and presents a small decrease afterwards. There is a high correlation between number of resets and transitions, over 0.7 for HSI and over 0.6 for P (Table 1).

Minimizing the number of resets is a known strategy to reduce the final length of the test suite. It can be obtained, e.g., by reducing the number of branches in the test suite (see Appendix 9.1 and [18] for more details). The SPY, H, and P methods clearly show the lowest numbers of resets, presenting small differences among them that can be explained by properties of their algorithms and sufficient conditions. HSI has a smaller number of resets in comparison with W, since the characterization set is usually greater than individual separation families. As a consequence, the concatenation with the transition cover will produce more test cases (resets).

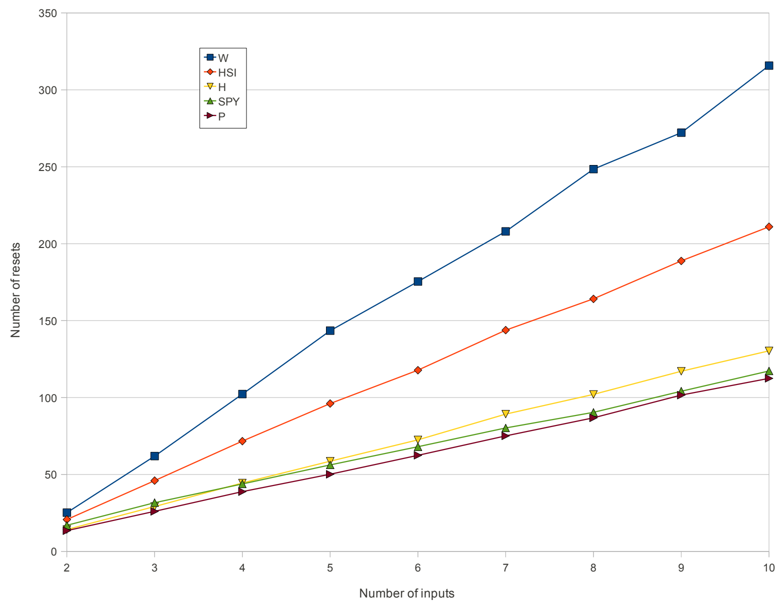


Fig. 6: Number of resets varying number of inputs.

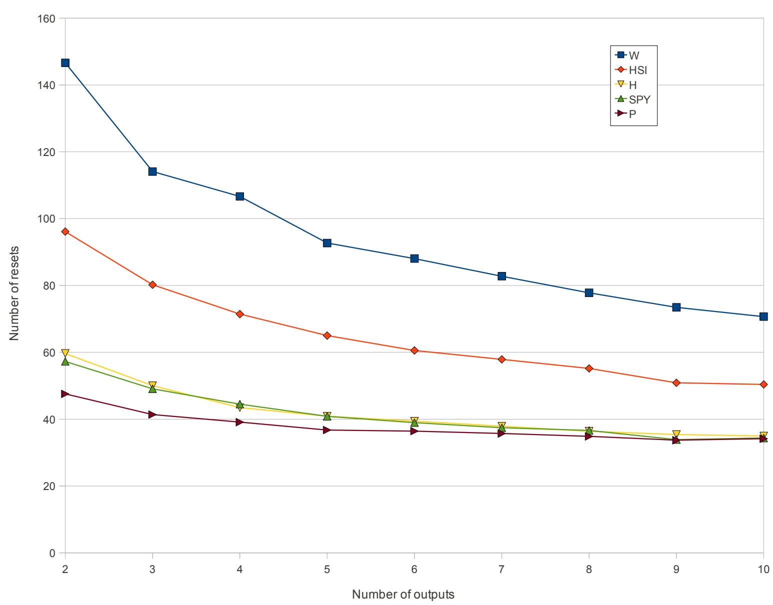


Fig. 7: Number of resets varying number of outputs.

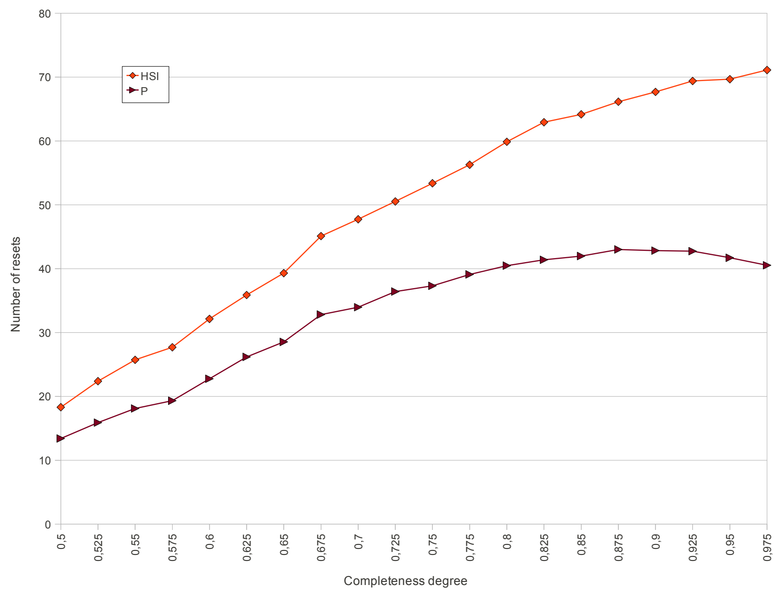


Fig. 8: Number of resets varying completeness degree.

5.1.2 Test Case Length

Table 2 shows the correlation between the average test case length and states, inputs, outputs, and transitions for each method. The results are discussed along with the following graphs.

Tab. 2: Correlation between average test case length and states, inputs, outputs, and transitions.

TC Len	States			Inputs			Outputs			Transitions		
	PCC	SCC	KCC	PCC	SCC	KCC	PCC	SCC	KCC	PCC	SCC	KCC
W	.89	.90	.75	-.80	-.92	-.80	-.36	-.34	-.25	-	-	-
HSI	.89	.90	.76	-.80	-.91	-.77	-.16	-.19	-.13	-.72	-.76	-.57
SPY	.77	.78	.61	-.49	-.47	-.34	-.35	-.36	-.26	-	-	-
H	.88	.90	.74	-.78	-.82	-.66	-.30	-.32	-.23	-	-	-
P	.90	.92	.78	-.77	-.79	-.63	-.63	-.63	-.47	-.54	-.54	-.38

Figures 9, 10, and 11 show how the minimum, average, and maximum test case lengths, respectively, vary in function of the number of states. Although a wide range of states was used, the variation in the test case length was not high. The minimum test case length ranges from 2 to 3.1; the average test case length ranges from 3.1 to 6.2; and the maximum test case length ranges from 3.8 to 12.3. There is a high positive correlation between the average test case length and the number of states (over 0.61 for the three coefficients in Table 2). This correlation is slightly lower with the SPY method.

The increase in the number of states also increases the test case lengths. As the FSMs have more states, longer sequences are necessary to reach and verify these states. However, this increase was not so accentuated. The HSI and W methods have the shortest test cases in the minimum, average, and maximum graphs. Values minL_H, avgL_H, and maxL_H place an intermediate position for the H method. For minimum and average test case lengths, SPY and P have similar results. For maximum lengths, SPY shows the longest test cases with constant difference of over two inputs symbols w.r.t. the P method.

Figures 12, 13, and 14 show how the minimum, average, and maximum test case lengths, respectively, vary in function of the number of inputs. In almost all methods, the test case length tends to slightly decrease with the growth of the number of inputs. As there are more transitions defined for each state, it is likely that shorter sequences can be identified to reach states from s_0. The only growth of test case length in function of the number of inputs is presented in Figure 14 in which the SPY method has an increase in the maximum test case length. For number of inputs, the following order $\mathrm{L}_{SPY} > \mathrm{L}_P > \mathrm{L}_H > \mathrm{L}_W > \mathrm{L}_{HSI}$ is observed for minimum, average, and maximum test case length. There is a negative correlation between test case length and number of inputs for almost all methods (under -0.63 for the three coefficients in Table 2), less SPY that presents a medium negative correlation. As the number of inputs increases (with fixed states and outputs), it is more likely to have shorter separation sequences and then, shorter test cases.

Figures 15, 16, and 17 show how the minimum, average, and maximum test case lengths, respectively, vary in function of the number of outputs. The average test case length has an initial reduction until four outputs and is constant afterwards. The increase of outputs tends to shorten the test cases since shorter separation sequences are more likely to be identified. However, after a given number of outputs, shorter separation sequences cannot be found. This explains the constant behavior after four outputs. In general, the behavior of minimum, average, and maximum test case lengths are similar to the presented for states and inputs. Notice that there is more variation in the correlations between test case length and number of outputs (Table 2). HSI presents a weak negative correlation near zero, while the W, SPY, and H methods present a low negative correlation (≈ -0.3) and P a medium one (≈ -0.6).

In general, W and HSI methods present similar test case lengths, with the shortest test cases among the methods. On average, HSI generates slightly shorter test case lengths than W. The SPY method presents the longest test cases in all configurations, followed by P and H. Table 3 shows the pairwise comparison between the average test case length of each method using 3200 complete FSMs (varying states, inputs, and outputs). By the comparisons, the following order is observed

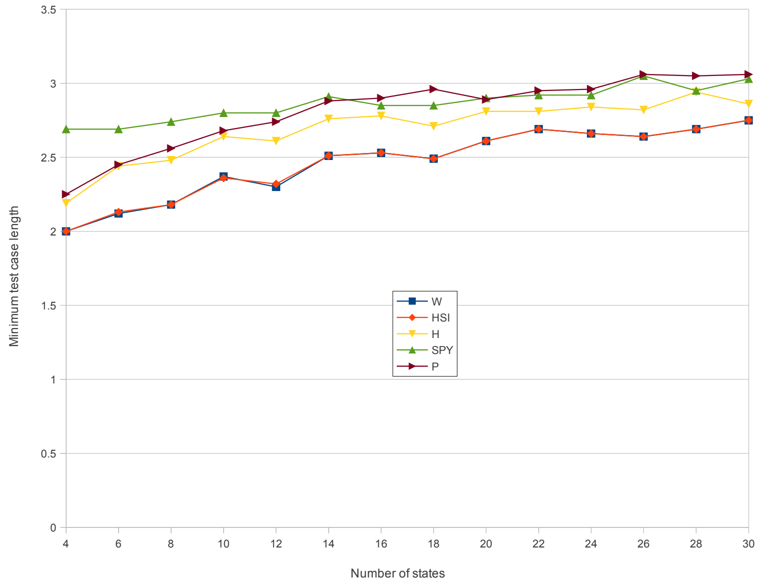


Fig. 9: Minimum test case length varying number of states.

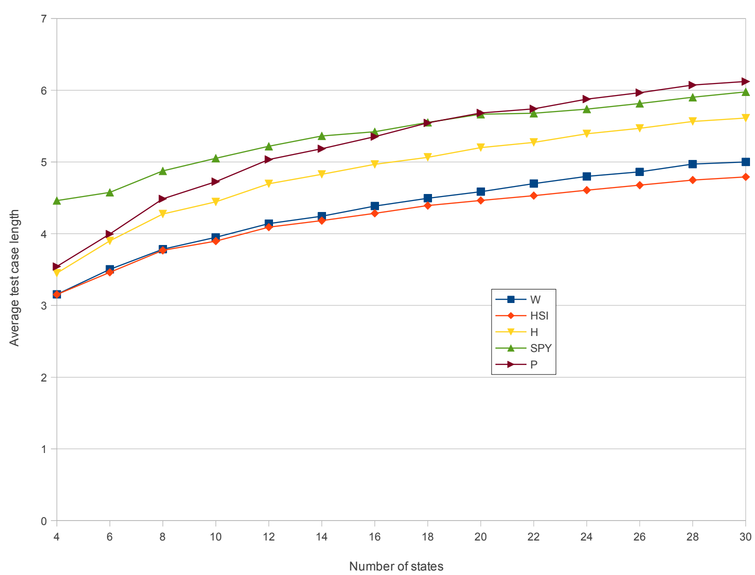


Fig. 10: Average test case length varying number of states.

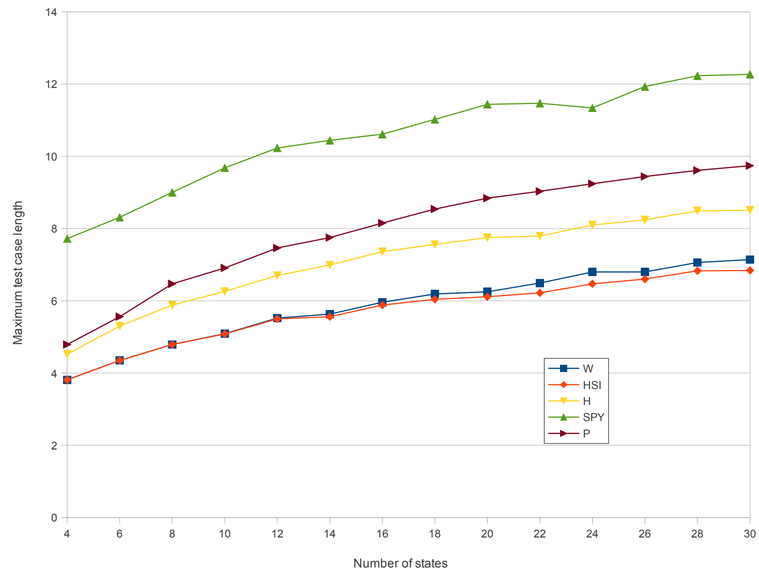


Fig. 11: Maximum test case length varying number of states.

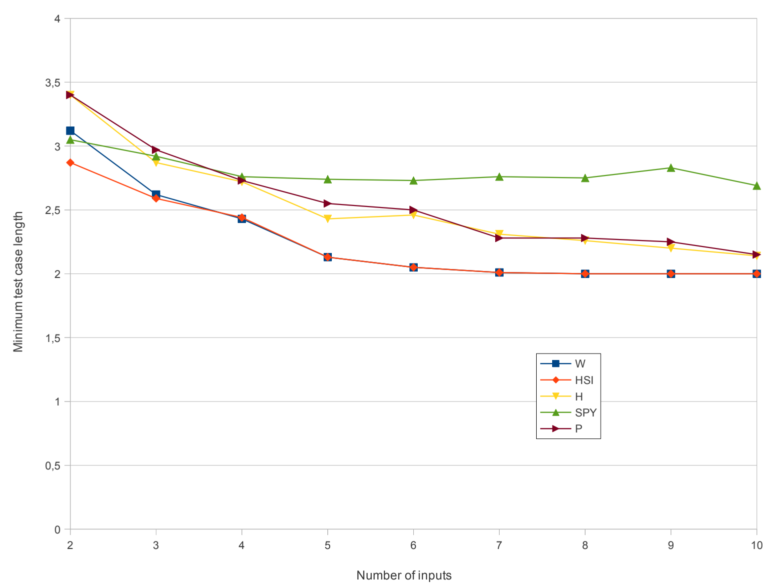


Fig. 12: Minimum test case length varying number of inputs.

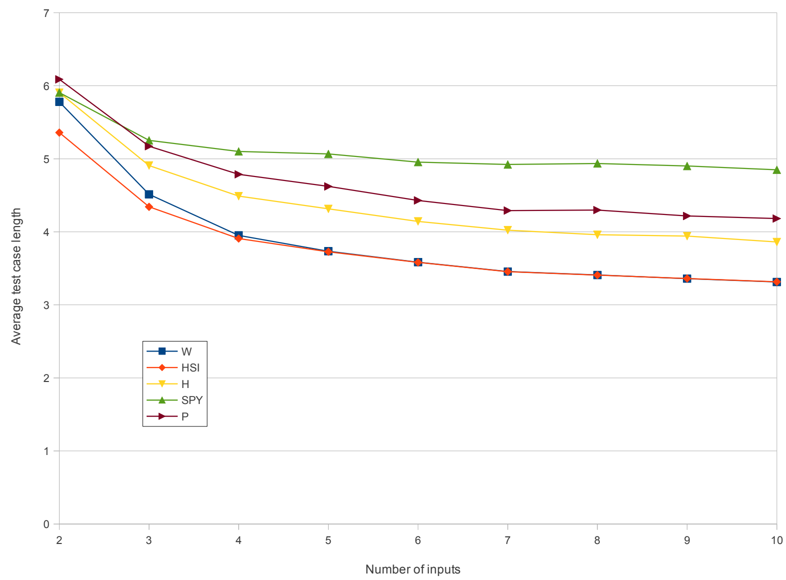


Fig. 13: Average test case length varying number of inputs.

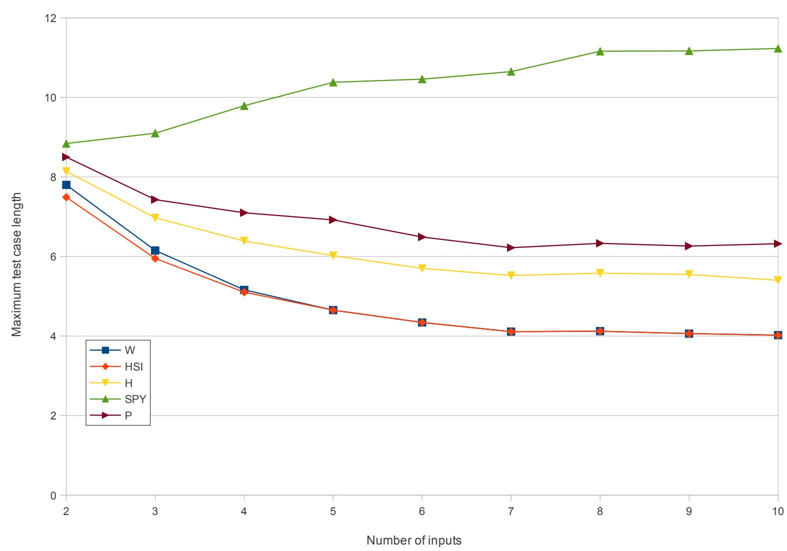


Fig. 14: Maximum test case length varying number of inputs.

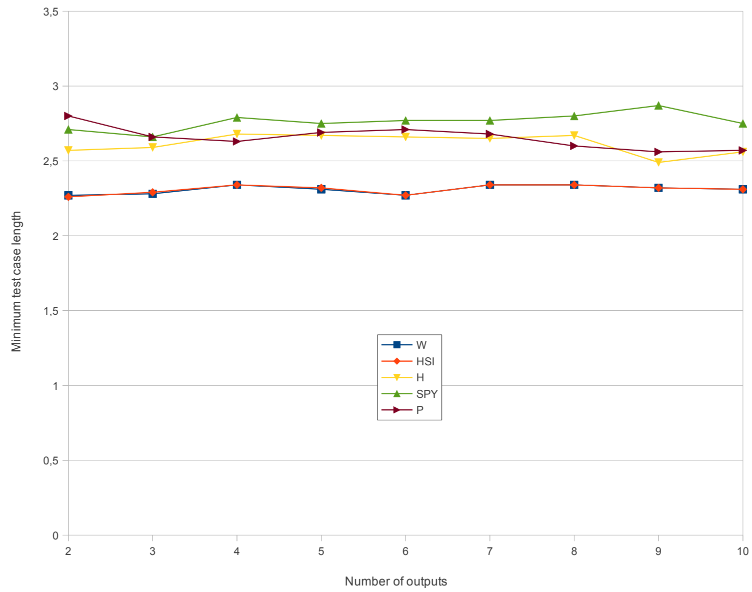


Fig. 15: Minimum test case length varying number of outputs.

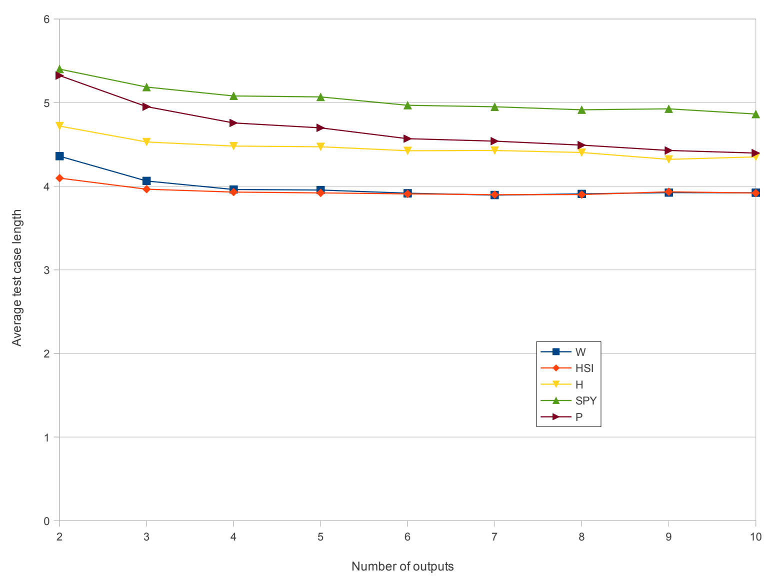


Fig. 16: Average test case length varying number of outputs.

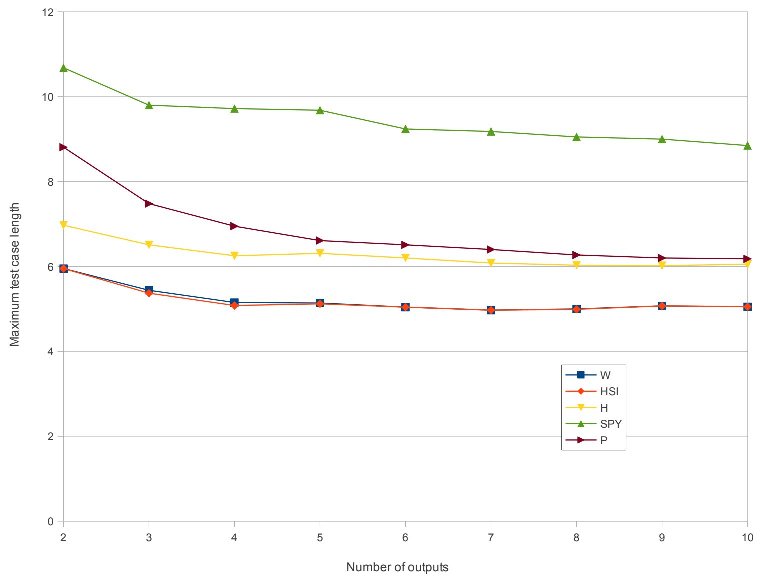


Fig. 17: Maximum test case length varying number of outputs.

and statistically supported avgL_{SPY} > avgL_P > avgL_H > avgL_W > avgL_{HSI}. The difference between W and HSI is minimum and they produce test cases that have virtually the same length on average. The SPY method has the longest test cases, though in practice this represents an average of ≈ 1.1 more input symbols than W and HSI.

Tab. 3: Test case length: pairwise comparison.

		W	HSI	SPY	H
HSI	tested order	W>HSI	-	-	-
	median diff	0.035	-	-	-
	mean diff	0.081	-	-	-
	SMD' d	0.141	-	-	-
	Hedges' g	0.141	-	-	-
SPY	tested order	SPY>W	SPY>HSI	-	-
	median diff	1.2	1.234	-	-
	mean diff	1.1	1.173	-	-
	SMD' d	1.88	2.19	-	-
	Hedges' g	1.88	2.19	-	-
H	tested order	H>W	H>HSI	SPY>H	-
	median diff	0.544	0.579	0.655	-
	mean diff	0.507	0.588	0.585	-
	SMD' d	0.801	0.997	0.987	-
	Hedges' g	0.801	0.997	0.987	-
P	tested order	P>W	P>HSI	SPY>P	P>H
	median diff	0.817	0.852	0.381	0.273
	mean diff	0.812	0.893	0.280	0.304
	SMD' d	1.196	1.398	0.436	0.442
	Hedges' g	1.196	1.398	0.436	0.442

Figure 18, 19, and 20 shows how the minimum, average, and maximum test case lengths vary in function of the number of transitions (completeness degree). The test case length shows some variation but in general it is a constant behavior around 3.7 and 5.4. As in complete FSMs, HSI

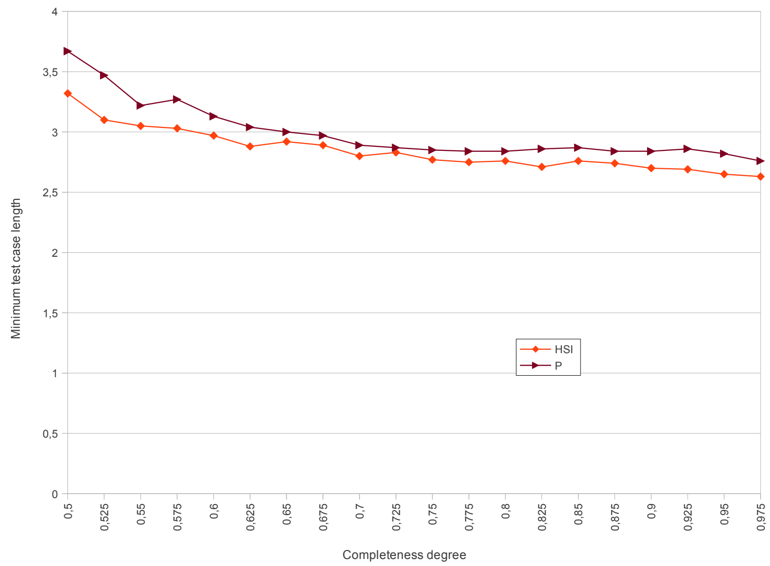


Fig. 18: Minimum test case length varying completeness degree.

has shorter test case lengths than P. In average test case lengths, the difference between HSI and P remains constant when varying the completeness degree. A medium negative correlation between test case length and transitions is observed for HSI (≈ -0.7) and P (≈ -0.5). As there are more transitions, it is more likely to find shorther paths to states and, as a consequence, shorter test cases.

Table 4 shows the differences between test case length in partial and complete FSMs. The statistical significance was calculated using Wilcoxon test with a p-value $< 2.2e^{-16}$. Although the P method has longer test cases than HSI in partial FSMs, the mean and median difference is smaller than in complete machines.

Tab. 4: Test case length: partial and complete FSMs.

	tested order	median diff	mean diff
partial FSMs	P>HSI	0.5809	0.5689
complete FSMs	P>HSI	0.8529	0.8936

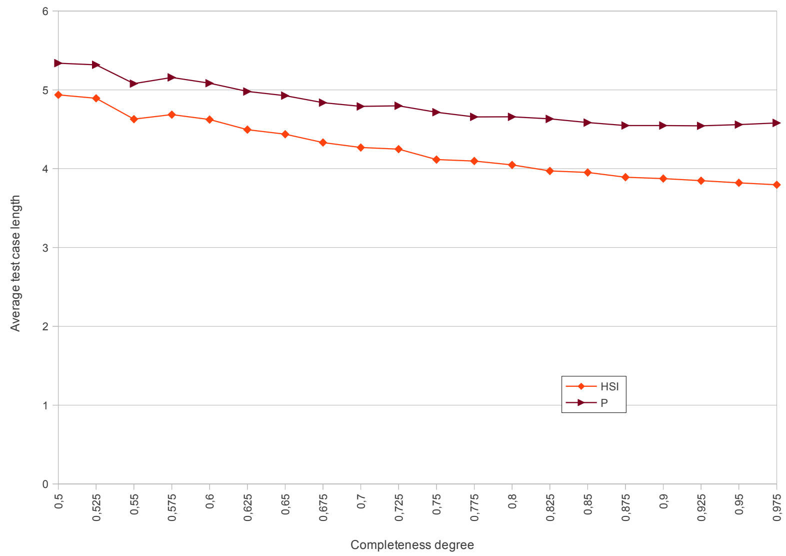


Fig. 19: Average test case length varying completeness degree.

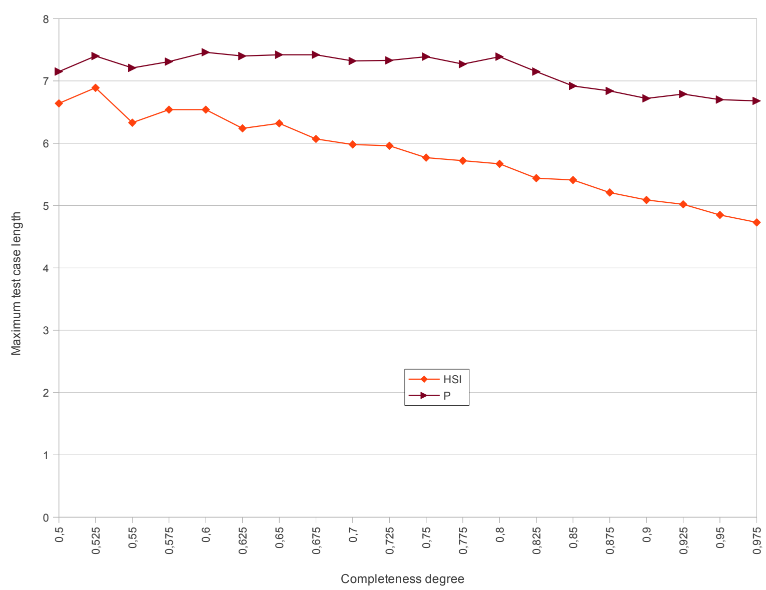


Fig. 20: Maximum test case length varying completeness degree.

5.2 RQ2: Test Suite Length

This section presents graphs and observations about the overall cost of the methods which is measured by the test suite length. Table 5 shows the correlation between the test suite length and states, inputs, outputs, and transitions for each method. The results are discussed along with the following graphs.

Tab. 5: Correlation between test suite length and states, inputs, outputs, and transitions.

TS len	States			Inputs			Outputs			Transitions		
	PCC	SCC	KCC	PCC	SCC	KCC	PCC	SCC	KCC	PCC	SCC	KCC
W	.96	.98	.91	.91	.92	.80	-.74	-.73	-.57	-	-	-
HSI	.98	.99	.94	.97	.97	.89	-.83	-.85	-.69	.87	.88	.71
SPY	.98	.99	.93	.97	.97	.89	-.79	-.80	-.64	-	-	-
H	.98	.99	.93	.96	.96	.87	-.77	-.77	-.61	-	-	-
P	.98	.99	.93	.97	.97	.89	-.75	-.75	-.59	.79	.78	.60

5.2.1 Number of States

Figure 21 shows how the test suite length varies in function of the number of states. We observed that the test suite length grows as the number of states increases. This is shown by a strong positive correlation between the number of states and the test suite length (over 0.91 for the three coefficients in Table 5). We also observed the order $|TS_{W}| > |TS_{HSI}| > |TS_{SPY}| > |TS_{H}| > |TS_{P}|$. The difference among SPY, H, and P is small from four to 20 states, but becomes larger and constant after 22 states.

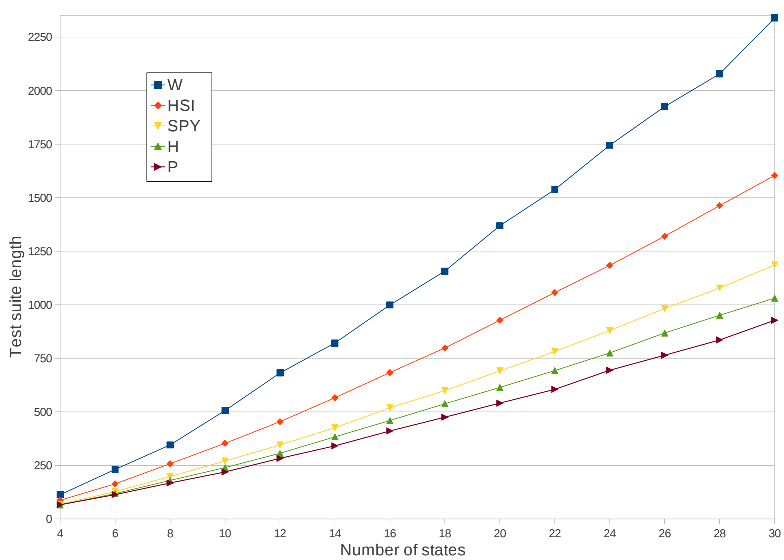


Fig. 21: Number of states.

We also analyzed the same data using the reduction ratio as the Y-axis. It varies until 12 states and presents a constant difference after 14 states. This constant difference represents a reduction of circa 30% for HSI, 50% for SPY, 55% for H, and 60% for P, in comparison with the test suites generated by the W method.

Figure 22 shows boxplots for each method in three configurations (four, 18, and 30 states); notice that the range of boxplots for FSMs with different numbers of states varies. In the configuration with four states, the recent methods present shorter test suites than W and HSI. However, the

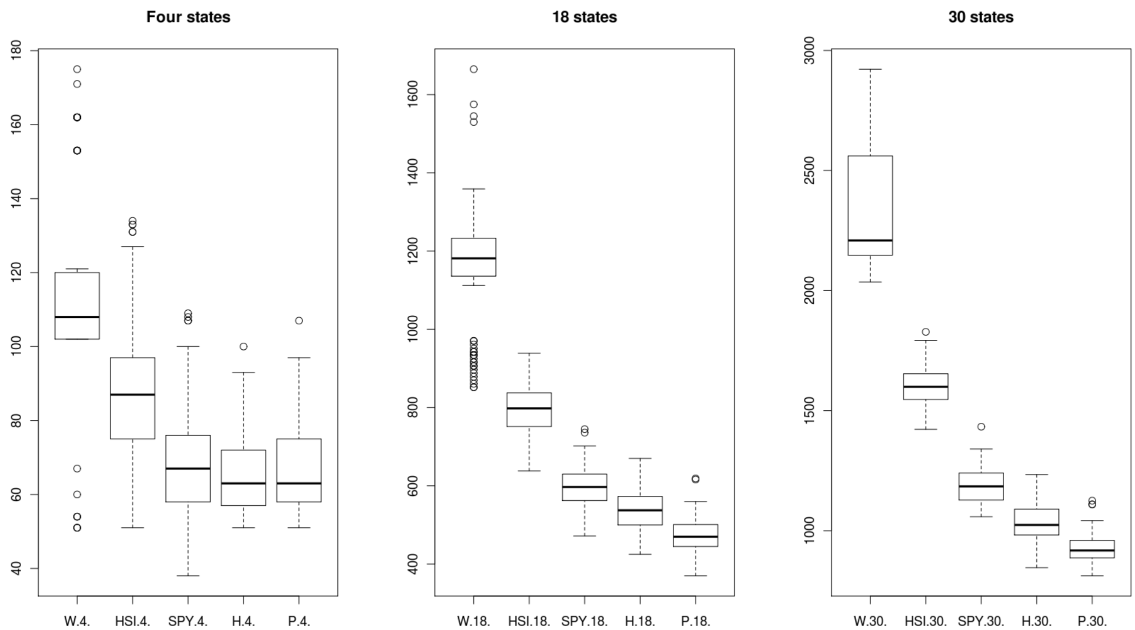


Fig. 22: Boxplots for number of states (four, 18, and 30 states).

interquartile ranges of SPY, H, and P are quite similar. These method performances are to some extent discriminated with more states (18 and 30). Moreover, the method distributions seem to be more symmetric with the increase of states, except for the skewed W distribution in 30 states. The W method also shows the highest number of outliers for the presented distributions.

5.2.2 Number of Inputs

Figure 23 shows how the test suite length varies in function of the number of inputs. We observed that the test suite length grows as the number of inputs increases. This is shown by a strong positive correlation observed between the number of inputs and the test suite length (over 0.8 for the three coefficients in Table 5). The same order $|TS_W| > |TS_{HSI}| > |TS_{SPY}| > |TS_H| > |TS_P|$ is observed.

Concerning the reduction ratio, the variation in the number of inputs presents a behavior similar to that of the number of states. However, the reduction ratio was less constant since a shorter range of values was considered.

Figure 24 shows boxplots for each method in three configurations (two, six, and ten inputs). Unlike Figure 22, an increase in the number of inputs preserves the W distribution asymmetric (positive skewness in the three configurations). W and HSI are to some extent more separated from recent methods with the increase of inputs, though their whiskers have reached the range of recent methods. Regarding to the median, P presents the best performance followed by H and SPY. In the recent methods, there is a reasonable difference between the interquartile ranges of SPY and P, which is not valid for SPY and H, and H and P.

5.2.3 Number of Outputs

Figure 25 shows how the test suite length varies in function of the number of outputs. We observed that the test suite length reduces as the number of outputs increases. This is shown by a negative correlation between the number of outputs and the test suite length (under -0.57 for the three coefficients in Table 5). The reduction is higher until four outputs and stabilizes after five outputs. From five to ten outputs, the SPY, H, and P methods produce similar test suite lengths. With more outputs, the distinguishing sequences tend to be shorter, reducing the choices for *on-the-fly* methods. In this case, the same distinguishing sequences are chosen by the methods whose test suites have similar lengths.

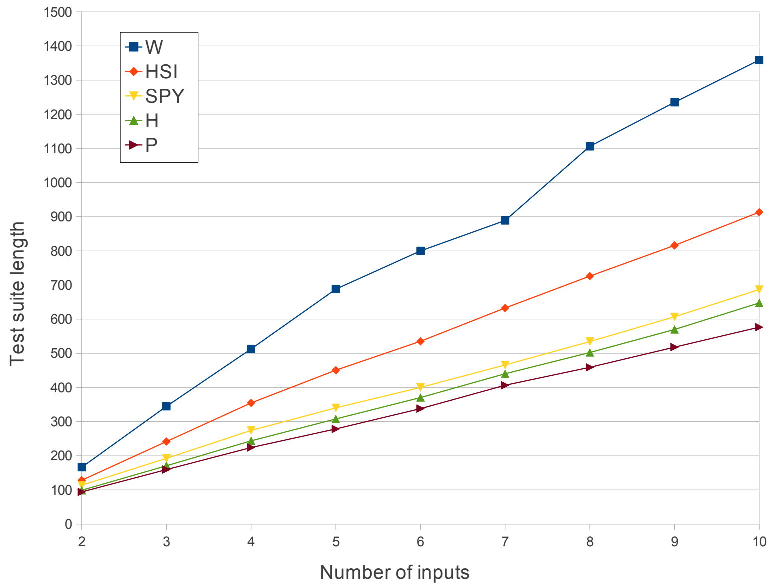


Fig. 23: Number of inputs.

The reduction is also not constant, though the HSI method keeps it around 30%. On the other hand, the reduction of P decreases to around 50%. H and SPY also present a similar behavior. These results have confirmed the intuition that with more outputs, shorter sequences can be found for distinguish states.

Figure 26 shows boxplots for each method in three configurations (two, six, and ten outputs). The tendency to shorter test suites with the increase in the number of outputs can also be observed. The W distribution presents different shapes, as in Figure 22. A large number of outliers can also be noticed with six outputs. The SPY, H, and P methods are initially discriminated by interquartile ranges with two outputs. However, these differences are reduced with more outputs, e.g., the H and P methods present similar boxplots with ten outputs.

5.2.4 Completeness Degree

Figure 27 shows how the test suite length varies in function of the number of transitions. There is a high positive correlation number of transitions and test suite length for partial FSMs (over 0.6 for the three coefficients in Table 5). This is also expected since there are more transitions to be verified and more test cases are required. While the growth of the test suite length until 27 (0.68) transitions was expected, this is not observed after 28 (0.70) transitions. The P method presents a constant behavior and some reduction at the end (39 transitions). This observation contradicts previous results with complete FSMs, in which an increasing number of transitions (by adding more inputs or states) implies a greater test suite length.

Figure 28 shows boxplots for each method in three configurations (completeness degree 0.5, 0.75, and 0.98). The difference between HSI and P is clearer with the increase in the completeness degree. At the lowest degree 0.5, the significant difference between HSI and P found in complete FSMs is not observed in partial FSMs, and the P boxplot is almost completely within the HSI whiskers range.

Figure 29 depicts the reduction ratio $|TS_P|/|TS_{HSI}|$ for partial FSMs (solid black line). A dashed gray line has been added representing the average for the same ratio calculated using complete FSMs with ten states, four inputs, and four outputs. As observed, the $|TS_P|/|TS_{HSI}|$ ratio is different when comparing complete and partial FSMs, suggesting that the difference w.r.t. test suite length between two methods tends to be smaller when applied to partial FSMs. Therefore, previous results for complete FSMs cannot be generalized directly to partial machines.

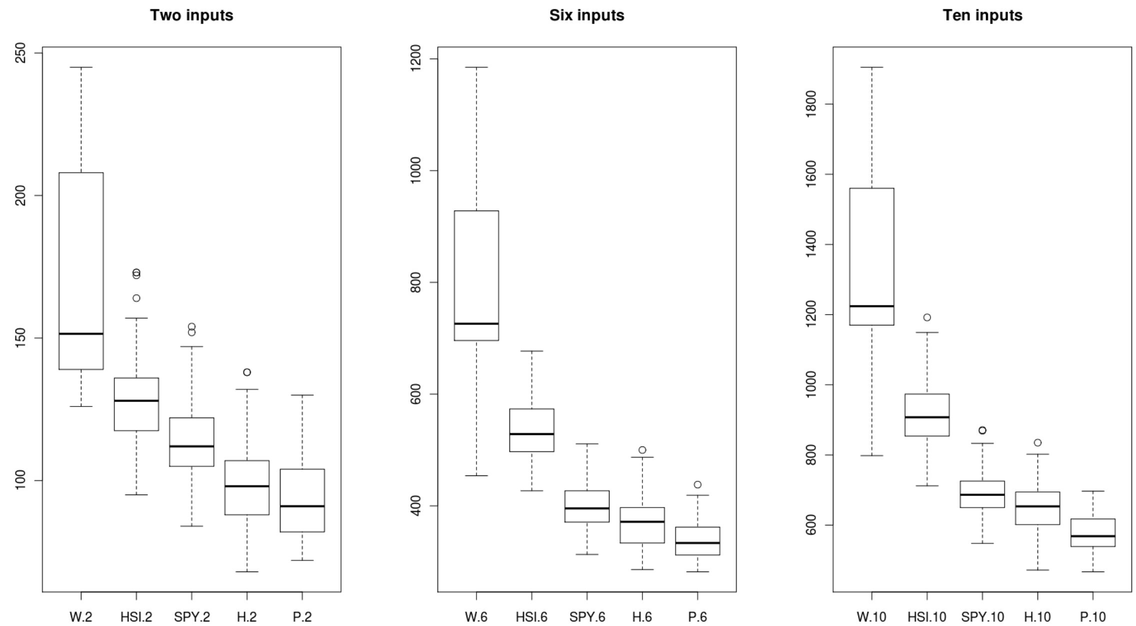


Fig. 24: Boxplots for number of inputs (two, six, and ten inputs).

Figure 30 shows boxplots for the reduction ratios analyzed in Figure 29 for complete and partial FSMs. It suggests that the reduction ratio $|TS_P|/|TS_{HSI}|$ in complete FSMs is different when the HSI and P methods are applied in partial machines. The median for complete FSMs is 0.62 and 0.76 for partial machines.

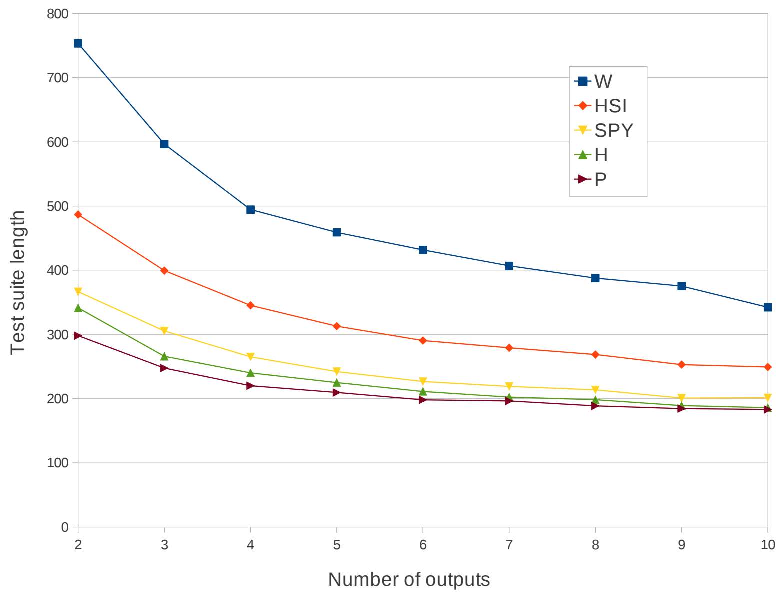


Fig. 25: Number of outputs.

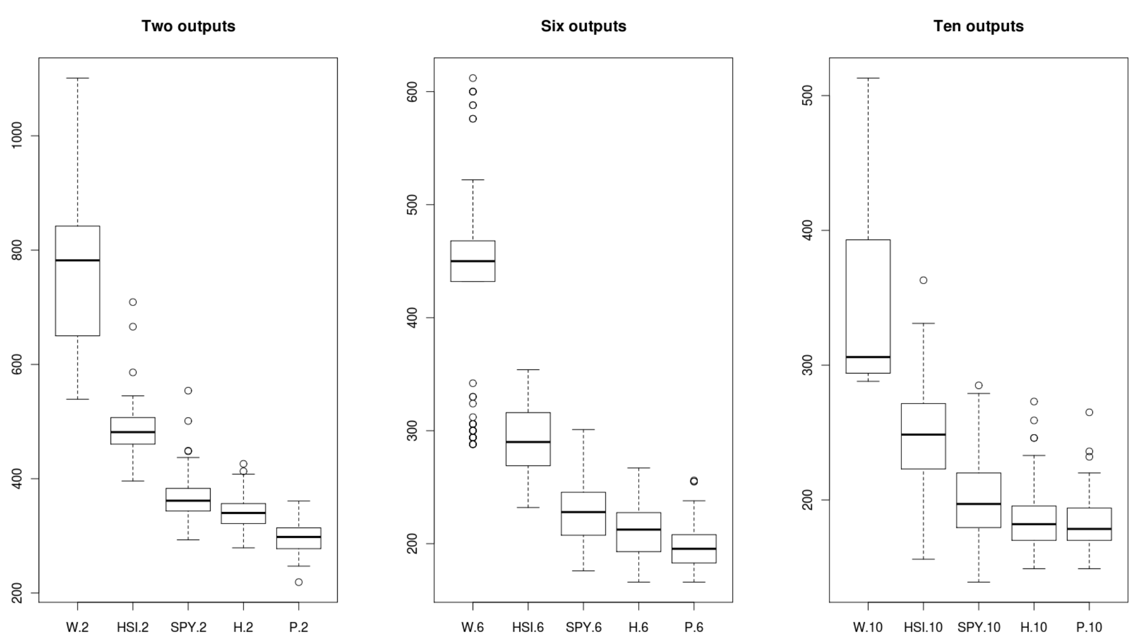


Fig. 26: Boxplots for number of outputs (two, six, and ten outputs).

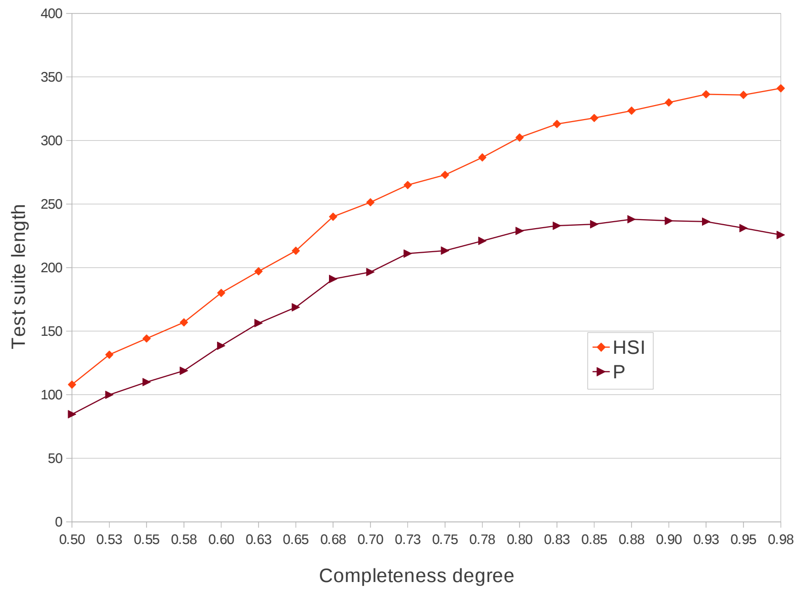


Fig. 27: Completeness degree (test suite length).

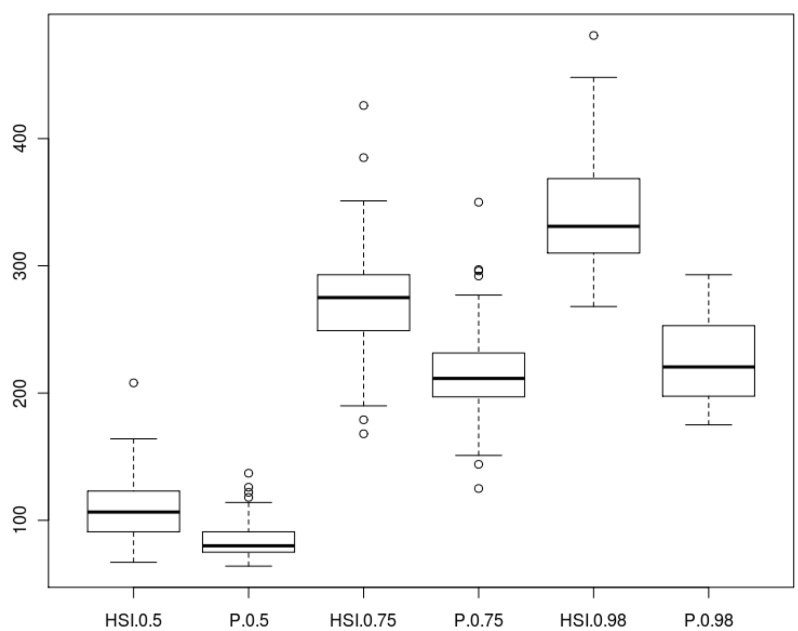


Fig. 28: Boxplots for completeness degree (0.5, 0.75, and 0.98).

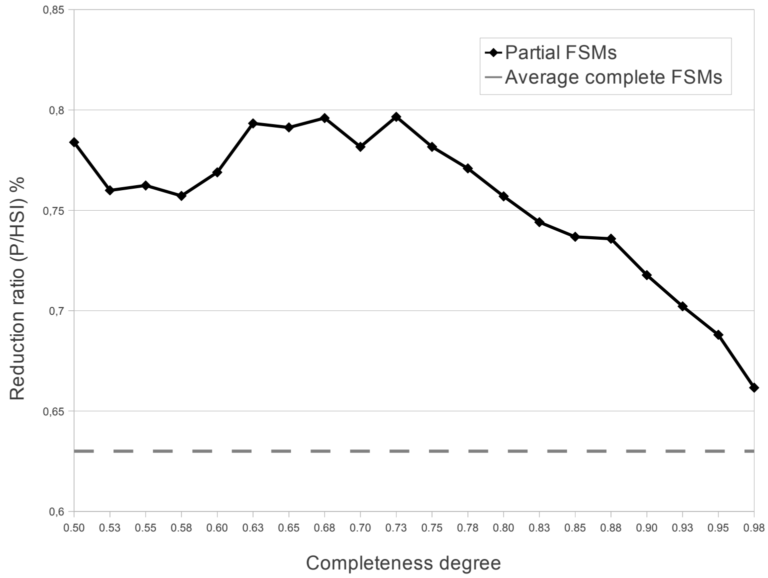


Fig. 29: Completeness degree (reduction ratio).

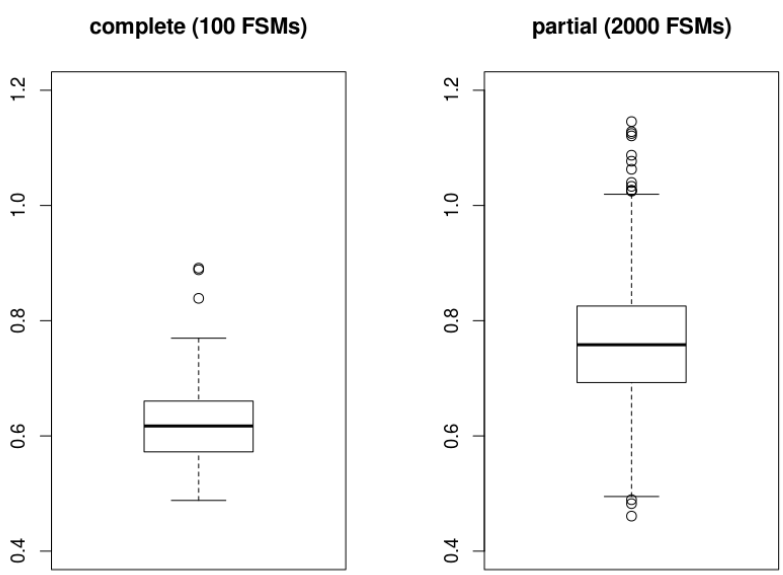


Fig. 30: Boxplots for the reduction ratio (completeness degree).

5.2.5 General Results

Table 6 shows the pairwise comparison of the test suite length for each method using 3200 complete FSMs (varying states, inputs, and outputs). The standardized effect sizes d and g are not defined when comparing the methods with W because this method was used to calculate the reduction ratio. Given that two methods μ_1 and μ_2, the notation $\mu_1 >_{a(b)} \mu_2$ means that μ_1 has test suite lengths longer than μ_2, a and b represent the median difference and the mean difference, respectively. Thus, we can summarize this table with the following expression $W >_{0.3(0.28)} HSI >_{0.17(0.16)} SPY >_{0.05(0.04)} H >_{0.04(0.03)} P$. Notice that the standardized effect size measures SMD' d and Hedges' g are provided as an alternative measure. Although there is no reference values for them, the values presented here can be used in future research. For instance, a reasonable improvement from SPY over HSI (median difference of 0.17) has values for d and g of over 1.0, while small improvements (e.g., 0.04 for P over H) have values of around 0.3.

Tab. 6: Test suite length (reduction ratio): pairwise comparison.

		W	HSI	SPY	H
HSI	tested order	W>HSI	-	-	-
	median diff	0.30	-	-	-
	mean diff	0.28	-	-	-
	SMD' d	-	-	-	-
	Hedges' g	-	-	-	-
SPY	tested order	W>SPY	HSI>SPY	-	-
	median diff	0.47	0.17	-	-
	mean diff	0.45	0.16	-	-
	SMD' d	-	1.11	-	-
	Hedges' g	-	1.11	-	-
H	tested order	W>H	HSI>H	SPY>H	-
	median diff	0.52	0.22	0.05	-
	mean diff	0.50	0.21	0.05	-
	SMD' d	-	1.51	0.40	-
	Hedges' g	-	1.51	0.40	-
P	tested order	W>P	HSI>P	SPY>P	H>P
	median diff	0.56	0.26	0.09	0.04
	mean diff	0.54	0.25	0.08	0.03
	SMD' d	-	1.78	0.69	0.30
	Hedges' g	-	1.78	0.69	0.30

Table 7 shows the differences between test suite length in partial and complete FSMs. It shows that reduction is clearly smaller when the methods are applied to partial FSMs.

Tab. 7: Test suite length: partial and complete FSMs.

	tested order	median diff	mean diff
partial FSMs	HSI>P	0.24	0.24
complete FSMs	HSI>P	0.36	0.35

5.3 RQ3: Fault Detection

Let TS be a test suite, $\#tm$ be the total of mutants, $\#km$ be the number of mutants killed by TS, and $\#em$ be the number of equivalent mutants, the *fault detection ratio* (FDR) is calculated by $\mathrm{FDR_{TS}} = \#km/(\#tm - \#em)$. This metric is also called mutation score. Further information on mutants and their relation with states, inputs, outputs, and transitions can be found in Appendix 9.2. Table 8 shows the correlation between the FDR and states, inputs, outputs, and transitions for each method. The results are discussed along with the following graphs.

Tab. 8: Correlation between the FDR and states, inputs, outputs, and transitions.

FDR	States			Inputs			Outputs			Transitions		
	PCC	SCC	KCC	PCC	SCC	KCC	PCC	SCC	KCC	PCC	SCC	KCC
W	.87	.97	.88	-.78	-.92	-.79	.07	.10	.10	-	-	-
HSI	.88	.99	.96	-.82	-.97	-.90	.66	.67	.51	-.79	-.82	-.64
SPY	.86	.99	.93	-.89	-.93	-.80	-.43	-.43	-.31	-	-	-
H	.87	.99	.96	-.86	-.96	-.87	-.36	-.34	-.25	-	-	-
P	.86	.99	.95	-.88	-.97	-.90	-.69	-.68	-.52	-.61	-.63	-.46

5.3.1 Number of States

Figure 31 shows the FDR in function of the number of states. All methods present a growth that is higher until 12 states and smaller afterwards. This is shown by a strong positive correlation between the number of states and the FDR for all methods (over 0.8 for the three coefficients in Table 8). All FDRs are superior to 0.9 for more than ten states. SPY has the highest FDR and HSI the lowest one. The P, H, and W methods present similar FDRs between HSI and SPY.

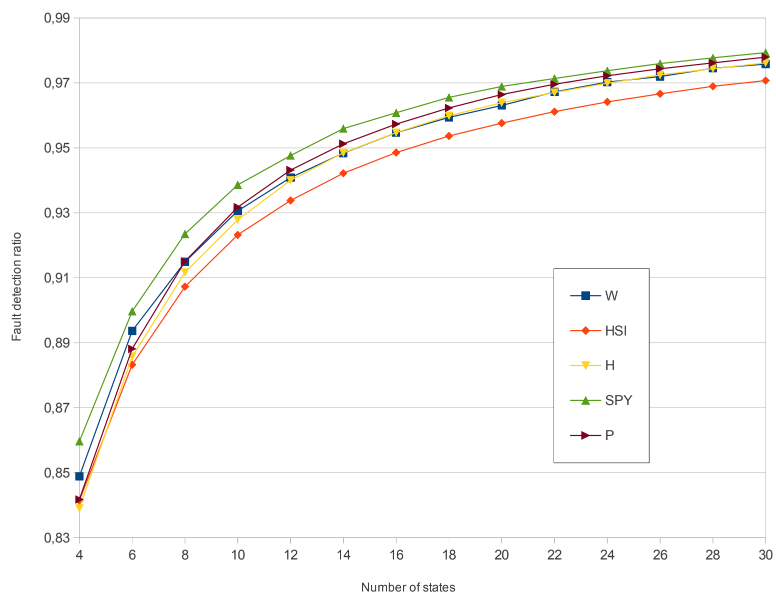


Fig. 31: FDR varying the number of states.

5.3.2 Number of Inputs

Figure 32 shows the FDR in function of the number of inputs. Here, the FDR's behavior is opposite of the number of states, though it is always superior to 0.9. The FDR tends to reduce with greater number of inputs. This is shown by a high negative correlation between the number of inputs and the FDR (under -0.79 for the three coefficients in Table 8).For two inputs, W and P present the highest FDRs, but after three inputs the FDR tends to be similar to the number of states.

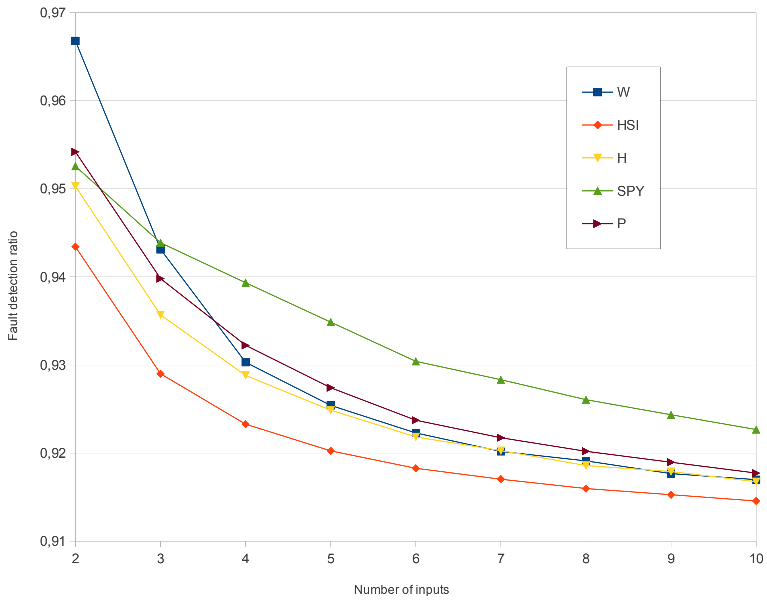


Fig. 32: FDR varying the number of inputs.

5.3.3 Number of Outputs

Figure 33 shows the FDR in function of the number of outputs. FDR shows much more variation when varying the number of outputs. When analyzing the correlation between the number of outputs and the FDR in Table 8, no conclusion can be drawn.The recent methods (SPY, H, and P) show a similar decrease of the FDR. On the other hand, W and HSI methods have a growth after three states. In the W method, basically there is no correlation between these variables (values close to0). HSI presents a moderate positive correlation, while the SPY, H, and P methods have a moderate negative correlation.

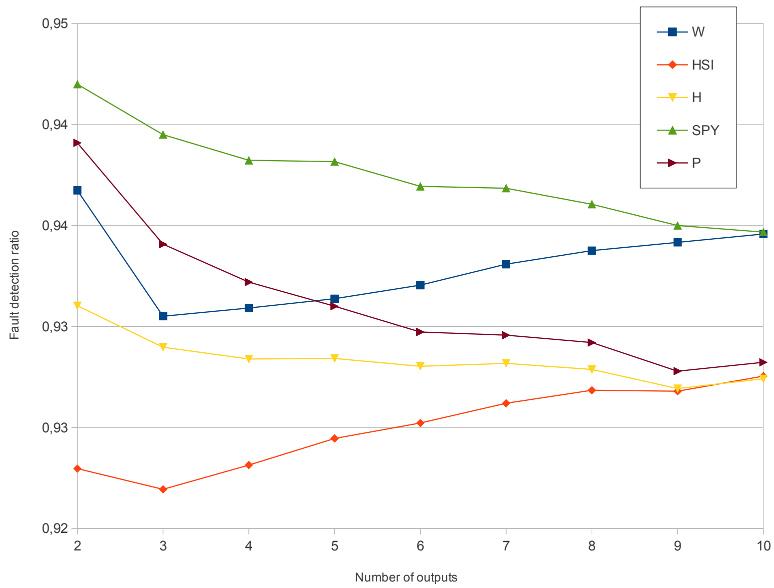


Fig. 33: FDR varying the number of outputs.

5.3.4 Completeness Degree

Figure 34 shows the FDR in function of the number of transitions (completeness degree). There is some variation in the FDR from 0.5 to 0.65 and a regular reduction after 0.68. In general, there is a negative correlation between the FDR and the number of transitions (under −0.46 for the three coefficients in Table 8). The P method has the highest FDR in partial FSMs and the HSI the lowest one.

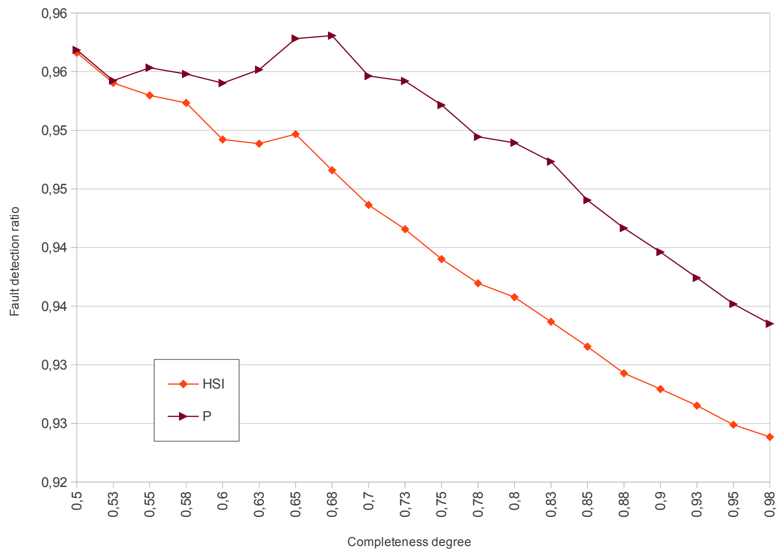


Fig. 34: FDR varying the number of transitions.

5.4 General Results

Table 9 shows the pairwise comparison of the FDR for each method using 3200 complete FSMs (varying states, inputs, and outputs). The statistical significance was calculated using Wilcoxon test with a p-value$<$ 0.0001. By using the pairwise comparison, the following order is observed $\mathrm{FDR}_{SPY} > \mathrm{FDR}_P > \mathrm{FDR}_W > \mathrm{FDR}_H > \mathrm{FDR}_{HSI}$.

All methods have a high average FDR in the interval $[0.9295, 0.9418]$. It means that most of the faults inserted by the mutation operators, even with a wrong estimation (-1) of number of states, are detected by n-complete test suites. Although it is statistically shown that the methods have different FDRs, the differences between them are quite small. For instance, the difference between the two methods in the limits, SPY and HSI, is around 0.01. In other words, SPY detects an average of 1% more faults (mutants) than HSI.

These differences between the methods' FDRs are relevant and can represent many undetected faults since the universe of faults is huge. For example, an average of $484,871$ mutants were produced for FSMs with 30 states, four inputs, and four outputs. This means that it is likely that by using SPY instead of HSI, $4,849$ more faults can be detected.

Table 10 shows the differences between the FDR in partial and complete FSMs. In partial FSMs, the P method also has a higher FDR than HSI. However, the median and mean differences between the methods are greater in partial machines. The average FDRs are also higher (0.94 for HSI and 0.949 for P) than in complete FSMs (0.929 for HSI and 0.936 for P).

Tab. 9: Fault detection ratio (FDR): pairwise comparison.

		W	HSI	SPY	H
HSI	tested order	W>HSI	-	-	-
	median diff	0.007	-	-	-
	mean diff	0.007	-	-	-
	SMD' d	0.27	-	-	-
	Hedges' g	0.27	-	-	-
SPY	tested order	SPY>W	SPY>HSI	-	-
	median diff	0.006	0.01	-	-
	mean diff	0.005	0.01	-	-
	SMD' d	0.20	0.49	-	-
	Hedges' g	0.20	0.49	-	-
H	tested order	W>H	H>HSI	SPY>H	-
	median diff	0.003	0.003	0.01	-
	mean diff	0.002	0.004	0.008	-
	SMD' d	0.10	0.16	0.31	-
	Hedges' g	0.10	0.16	0.31	-
P	tested order	P>W	P>HSI	SPY>P	P>H
	median diff	0.0006	0.006	0.007	0.003
	mean diff	0.0002	0.006	0.005	0.002
	SMD' d	0.01	0.26	0.21	0.09
	Hedges' g	0.01	0.26	0.21	0.09

Tab. 10: FDR: partial and complete FSMs.

	tested order	median diff	mean diff	SMD' d	Hedges' g
partial FSMs	P>HSI	0.009	0.008	0.73	0.73
complete FSMs	P>HSI	0.006	0.006	0.26	0.26

5.5 RQ4: Correlation

In this section, we show the correlations between the aspects analyzed in the questions RQ1, RQ2, and RQ3. Hence, we analyzed the correlations between test case length (RQ1), number of resets (RQ1), test suite length (RQ2), and FDR (RQ3) for each method in both complete and partial FSMs.

Table 11 shows the correlations between the average test case length and the number of resets. In complete FSMS, we observe a weak positive correlation for all methods. In partial FSMs, there exists a moderate negative correlation, which is higher for HSI.

Tab. 11: Correlation between average test case length and number of resets.

complete			
Method	**PCC**	**SCC**	**KCC**
W	0.2489	0.2408	0.1971
HSI	0.2657	0.2417	0.1820
SPY	0.4091	0.3580	0.2562
H	0.2907	0.2365	0.1698
P	0.3782	0.3247	0.2491
partial			
HSI	-0.7026	-0.7271	-0.5307
P	-0.5656	-0.5614	-0.3996

Table 12 shows the correlations between the average test case length and the test suite length.

In complete FSMs, we observe a moderate positive correlation. On the other hand, a moderate negative correlation is observed for partial FSMs.

Tab. 12: Correlation between average test case length and test suite length.

Method	PCC	SCC	KCC
complete			
W	0.4025	0.3468	0.2850
HSI	0.4020	0.3316	0.2450
SPY	0.5010	0.4479	0.3248
H	0.4513	0.3525	0.2534
P	0.5479	0.4526	0.3443
partial			
HSI	-0.5858	-0.6061	-0.4206
P	-0.4174	-0.4038	-0.2743

Table 13 shows the correlations between the average test case length and the FDR. In complete FSMs, we observe a high positive correlation (over 0.7 for PCC and SCC and over 0.5 for KCC). In partial FSMs, this results is weaker, though a moderate positive correlation is still observed. These results provides an useful information in which n-complete test suites with longer test cases tends to detect more faults (for the applied operators).

Tab. 13: Correlation between average test case length and FDR.

Method	PCC	SCC	KCC
complete			
W	0.80	0.88	0.71
HSI	0.75	0.85	0.67
SPY	0.70	0.74	0.55
H	0.80	0.87	0.69
P	0.85	0.89	0.72
partial			
HSI	0.68	0.75	0.54
P	0.38	0.41	0.28

Table 14 shows the correlations between the number of resets and the test suite length. There exists a strong positive correlation for all methods and in complete and partial FSMs (over 0.97). As there are more resets (test cases) in the n-complete test suite, the test suite length tends to be greater.

Tab. 14: Correlation between number of resets and test suite length.

Method	PCC	SCC	KCC
complete			
W	0.97	0.99	0.92
HSI	0.98	0.99	0.93
SPY	0.99	0.99	0.93
H	0.97	0.99	0.92
P	0.97	0.98	0.91
partial			
HSI	0.98	0.98	0.90
P	0.98	0.97	0.89

Table 15 shows the correlations between the number of resets and the FDR. Notice a moderate positive correlation in complete FSMs. Intuitively, adding more test cases (resets) will likely increase the FDR. However, this is not so evident for the observed results. In partial FSMs, we observe a negative correlation that is moderate for HSI and low for P.

Tab. 15: Correlation between number of resets and FDR.

complete			
Method	**PCC**	**SCC**	**KCC**
W	0.5141	0.4029	0.3230
HSI	0.6196	0.4053	0.2829
SPY	0.6239	0.4835	0.3571
H	0.5750	0.4154	0.3155
P	0.5434	0.4166	0.3409
partial			
HSI	-0.6790	-0.7044	-0.5011
P	-0.2741	-0.2915	-0.1914

Table 16 shows the correlations between the test suite length and the FDR. The results are quite similar to the correlation between number of resets and FDR (Table 15). More test cases do not necessarily guarantee that more faults will be detected.

Tab. 16: Correlation between test suite length and FDR.

complete			
Method	**PCC**	**SCC**	**KCC**
W	0.6093	0.4823	0.3673
HSI	0.6927	0.4735	0.3230
SPY	0.6675	0.5327	0.3989
H	0.6695	0.5058	0.3773
P	0.6531	0.5161	0.4086
partial			
HSI	-0.6065	-0.6356	-0.4390
P	-0.2024	-0.2215	-0.1415

6 Discussion

As previously described, the FSMs used in the experiments were randomly generated. In this case, the similarity between these FSMs and the ones used in practice is unclear. We aim to overcome this threat in a future work by replicating the experiment using real-world FSMs as subjects. Another issue is that a randomly generated FSM can represent a very rare and special case in which one of the methods does not perform well. Our solution is to use a high number of different FSMs for each configuration and calculate average measures, aiming to reduce the influence of this factor on the results.

As stated by Arcuri [28], there has been little reasearch on the impact of length of test cases. In this study, we provided a set of measures that can be used to shed light on this topic in the context of FSM-based testing. The recent methods tend to produce test suites with longer test cases, while traditional methods produce more (by the resets' data) and shorter test cases. This information can be useful for testers to support decision making during the MBT process. In general, shorter test cases are easier to execute and debug and can be more appropriate if they are executed by hand.

Traditional methods could be theoretically compared under some assumptions. However, if different algorithms were used to select characterization sets and separating families, there would be cases in which $|TS_W| < |TS_{HSI}|$. Concerning the recent methods, we can make naive comparisons based on sufficient conditions used by each method. Thus, the more general the sufficient conditions, the shorter the generated test suites tend to be. This is not always true since more aspects, like the strategy to select the distinguishing sequences, influence these methods. For instance, the SPY method generated longer test suites than the H method in the experiments, though SPY relies on more general sufficient conditions.

The results presented in this paper have some practical implications. There is a significant cost difference among the traditional and recent methods (around 50%). Even if we consider only recent methods, their reduction varies from 50 to 60%. This difference can be significant for models with many states, an SUT with an expensive test execution environment, or projects in which test sequences are executed by hand.

Differences between the fault detection ratio among the methods are quite close, though they are statistically different. A small percentage of undetected faults can represent a large number of faults since the fault domain is huge. Moreover, MBT has been applied in safety-critical software with special attention to embedded systems [29]. In this scenario, a small optimization of the fault detection ratio might be of high practical importance for safety-critical software [24].

The set of analyses about the methods' behavior in different configurations can support the choice of the best method to fit a given project. The tester can identify the tradeoffs between test case characteristics, test suite length, and fault detection ratio by analyzing the results presented. For instance, the tester may want a method that produces n-complete test suites with shorter test case lengths from partial FSMs. In this case, the HSI method would be the best choice. On the other hand, the SPY method would be better to detect more faults when the number of states in the implementation is likely wrong. Other decisions can be supported by the results presented in this work.

7 Related Work

Papers that introduce new methods provide some preliminary experimental results, usually comparing their methods with HSI [14, 18]. Analyzing the reduction ratio presented in Dorofeeva et al. [14] and Simao et al. [18], we could observe slight differences. Our results have shown small reductions for H and SPY methods compared with the HSI method. This fact can be explained by the use of different algorithms and heuristics to produce identification sets and different ranges of FSMs selected for the experiments.

Recent research towards experimental evaluation of FSM-based testing can be found in the literature [20, 11, 30]. Simao et al. [30] describe an experimental comparison among different coverage criteria for FSMs, such as state, transition, initialization fault, and transition fault coverage. Although the authors investigate how these criteria are related with n-complete test suites, no specific test generation method is compared.

Dorofeeva et al. [11] present an experimental evaluation of FSM methods, such as DS, W, HSI, Wp, UIO, and H. Their experiments focus on complete FSMs, analyzing mainly the number of transitions. They also consider methods that are not applicable for all minimal FSMs, such as DS [31] and UIO [32]. Our work can be seen as an extension of this study, considering new methods (SPY, P) and analyzing different dimensions (number of states, inputs, and outputs). Moreover, we also experimented the methods' performance in partial machines for HSI and P, varying the completeness degree. Our results have confirmed the order $|TS_W| > |TS_{HSI}| > |TS_H|$ described by Dorofeeva et al. [11].

Characteristics of test cases, such as the number of test cases and length, have been studied in the context of branch coverage [28] and specification based testing [33]. Arcuri [28] analyzes how the length of test cases influences the branch (all-edges) coverage, studying four search algorithms. A case study was conducted using six Java containers in which test cases are sequences of method calls. The author analyzes the task of finding a single and minimal test case to cover a difficult branch. He identifies that longer test sequences can have drastically better results to cover difficult test requirements.

Fraser and Gargantini [33] present a study on the effects of the test case length in a scenario of specification based testing of reactive systems. Their results show that longer test cases are not productive if minimization is applied afterwards. However, in most of the scenarios, fewer and longer test cases is more cost-effective than many short test cases. We observe a similar result in this study so that the recent methods (that produce fewer and longer test cases) detected more faults than traditional methods (many short test cases). A high positive correlation between test case length and fault detection ratio was also identified in complete FSMs and a moderate positive correlation in partial machines.

8 Conclusion and Future Work

This technical report has presented an experimental evaluation of test case generation methods for finite state machines (FSMs). The W, HSI, SPY, H, and P methods were compared considering number of states, inputs, outputs, and completeness degree. For each configuration, a set of FSMs was randomly generated. The methods were compared with respect to test case characteristics (number of resets and test case length), test suite length (cost), and fault detection ratio (effectiveness). On average, SPY has the longest test cases and HSI has the shortest ones. The recent methods resulted in shorter test suites compared with traditional methods. On average, the P method generated the shortest test suites, although the difference was smaller in comparison with H and SPY. Varying the number of states and inputs resulted in an increase in the test suite length. Increasing the number of outputs reduced the test suite length. With partial machines, we observed that the increase in the number of transitions does not necessarily increase the test suite length. Moreover, the reduction ratio between methods applied in complete FSMs cannot be generalized to partial ones. The reduction ratio for partial FSMs is smaller than that for complete machines. In the analyzed scenario, all methods showed a high average fault detection ratio (over 92%). The SPY and P methods had the highest fault detection and H and HSI had the lowest ratio. Different correlations were also identified along with the results.

As far as we know, there is no comparable work that provides a full comparison of FSM methods, including test case characteristics, test suite length, and fault detection ratio. More research on the behavior of existing methods for partial FSMs is necessary. Our results have shown that experimental evaluation on complete FSMs cannot be generalized to partial FSMs. The study of outliers can also shed light on FSM characteristics that increase/reduce the methods' performance. As future work, we plan to replicate this study, but focusing on partial and nondeterministic machines. Another line is the investigation on how separating sequences derived using different algorithms and strategies can impact on the test suites produced.

Acknowledgements

Andre Takeshi Endo is financially supported by FAPESP/Brazil (grant 2009/ 01486-9) and CAPES/ Brazil (grant 0332-11-9). Adenilso Simao is partially financially supported by CNPq/Brazil (grant 474152/2010-3). We would like to thank to Nina Yevtushenko and Rita Dorofeeva for the implementation of the H method and Jorge Cutigi for the implementation of the HSI method.

References

[1] I. K. El-Far and J. A. Whittaker, "Model-based software testing," in *Encyclopedia on Software Engineering.* Wiley, 2001, pp. 825–837.

[2] R. M. Hierons, K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Lüttgen, A. J. H. Simons, S. Vilkomir, M. R. Woodward, and H. Zedan, "Using formal specifications to support testing," *ACM Computing Surveys (CSUR)*, vol. 41, no. 2, pp. 1–76, 2009.

[3] M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

[4] W. Grieskamp, N. Kicillof, K. Stobie, and V. A. Braberman, "Model-based quality assurance of protocol documentation: tools and methodology," *Software Testing, Verification and Reliability*, vol. 21, no. 1, pp. 55–71, 2011.

[5] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines - a survey," *Proceedings of the IEEE*, vol. 84, no. 8, pp. 1090–1123, Aug 1996.

[6] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, *Model-based Testing of Reactive Systems*, 1st ed. Springer, 2005.

[7] E. F. Moore, "Gedanken-experiments on sequential machines," *Automata Studies, Annals of Mathematics Series*, no. 34, pp. 129–153, 1956.

[8] A. Gill, *Introduction to the theory of finite-state machines.* McGraw-Hill, 1962.

[9] A. Simao and A. Petrenko, "Fault coverage-driven incremental test generation," *Computer Journal*, vol. 53, pp. 1508–1522, 2010.

[10] ——, "Checking completeness of tests for finite state machines," *IEEE Transactions on Computers*, vol. 59, pp. 1023–1032, 2010.

[11] R. Dorofeeva et al., "FSM-based conformance testing methods: A survey annotated with experimental evaluation," *Information and Software Technology*, vol. 52, no. 12, pp. 1286 – 1297, 2010.

[12] R. M. Hierons and H. Ural, "Generating a checking sequence with a minimum number of reset transitions," *Automated Software Engineering Journal*, vol. 17, no. 3, pp. 217–250, Sep. 2010.

[13] L. L. C. Pedrosa and A. V. Moura, "Incremental testing of finite state machines," *Software Testing, Verification and Reliability*, pp. n/a–n/a, 2012. [Online]. Available: http://dx.doi.org/10.1002/stvr.1474

[14] R. Dorofeeva, K. El-Fakih, and N. Yevtushenko, "An improved conformance testing method," in *IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE).* Springer, 2005, pp. 204–218.

[15] M. P. Vasilevskii, "Failure diagnosis of automata," *Cybernetics and Systems Analysis*, vol. 9, pp. 653–665, 1973.

[16] T. S. Chow, "Testing software design modeled by finite-state machines," *IEEE Transactions on Software Engineering*, vol. 4, no. 3, pp. 178–187, 1978.

[17] G. Luo, A. Petrenko, and G. V. Bochmann, "Selecting test sequences for partially-specified nondeterministic finite state machines," in *International Workshop on Protocol Test Systems (IWPTS)*, 1995, pp. 95–110.

[18] A. Simao, A. Petrenko, and N. Yevtushenko, "Generating reduced tests for FSMs with extra states," in *IFIP WG 6.1 International Conference on Testing of Software and Communication Systems (TESTCOM).* Springer-Verlag, 2009, pp. 129–145.

[19] H. Ural, X. Wu, and F. Zhang, "On minimizing the lengths of checking sequences," *IEEE Transactions on Computers*, vol. 46, no. 1, pp. 93–99, Jan. 1997.

[20] R. Dorofeeva, K. El-Fakih, S. Maag, A. R.Cavalli, and N. Yevtushenko, "Experimental evaluation of FSM-based testing methods," in *International Conference on Software Engineering and Formal Methods (SEFM)*, 2005, pp. 23–32.

[21] A. T. Endo and A. Simao, "Experimental comparison of test case generation methods for finite state machines," in *The 8th Workshop on Advances in Model Based Testing (A-MOST 2012)*, 2012, pp. 549–558.

[22] Y. Jia and M. Harman, "Constructing subtle faults using higher order mutation testing," in *Eighth IEEE International Working Conference on Source Code Analysis and Manipulation*, 2008, pp. 249–258.

[23] A. Petrenko and N. Yevtushenko, "Testing from partial deterministic fsm specifications," *IEEE Transactions on Computers*, vol. 54, no. 9, Sep. 2005.

[24] V. B. Kampenes, T. Dybå, J. E. Hannay, and D. I. K. Sjøberg, "Systematic review: A systematic review of effect size in software engineering experiments," *Information & Software Technology*, vol. 49, no. 11-12, pp. 1073–1086, Nov. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2007.02.015

[25] T. Dybå, V. B. Kampenes, and D. I. K. Sjøberg, "A systematic review of statistical power in software engineering experiments," *Information & Software Technology*, vol. 48, no. 8, pp. 745–755, 2006.

[26] R-Project, "The R project for statistical computing," 2012. [Online]. Available: http://www.r-project.org/

[27] Gnuplot, "Gnuplot website," 2012. [Online]. Available: http://www.gnuplot.info/

[28] A. Arcuri, "Longer is better: On the role of test sequence length in software testing," *Software Testing, Verification, and Validation, 2008 International Conference on*, vol. 0, pp. 469–478, 2010.

[29] J. Zander, I. Schieferdecker, and P. Mosterman, *Model-Based Testing for Embedded Systems*, ser. Computational Analysis, Synthesis, and Design of Dynamic Systems. Taylor & Francis, 2011.

[30] A. Simao, A. Petrenko, and J. C. Maldonado, "Comparing finite state machine test coverage criteria," *IET Software*, vol. 3, no. 2, pp. 91–105, 2009.

[31] F. C. Hennie, "Fault detecting experiments for sequential circuits," in *Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, nov. 1964, pp. 95 –110.

[32] K. Sabnani and A. Dahbura, "A protocol test generation procedure," *Computer Networks and ISDN Systems*, vol. 15, no. 4, 1988.

[33] G. Fraser and A. Gargantini, "Experiments on the test case length in specification based test case generation." in *AST*, D. Dranidis, S. P. Masticola, and P. A. Strooper, Eds. IEEE, 2009, pp. 18–26.

[34] L. F. Mello Neto and A. Simao, "Test suite minimization based on FSM completeness sufficient conditions," in *9th IEEE Latin-American Test Workshop*, Puebla, Mexico, 2008, pp. 93–98.

9 Appendix

9.1 Reducing the number of branches

Consider the FSM presented in Figure 1 and the sequence $\alpha = ab$. When applied to the initial state, the sequence lead the machine to state s_2. Suppose that we want to distinguish s_2 from other states s_0, s_1, s_3, s_4. Table 17 presents sequences that might be used to perform this task. We present the outputs for each state, using all sequences with size from one to three.

Note that many sets of sequences might be used to distinguish s_2 from other states, such as $D_1 = \{a, b\}, D_2 = \{aa\}, D_3 = \{bb, a\}, D_4 = \{bab\}$, and $D_5 = \{a, bb, baa\}$. The sets $\alpha.D_i$, $1 \leq i \leq 5$ have the same properties, but different characteristics.

In traditional methods, distinguishing sets are produced previously, therefore it is not possible to choose the more appropriate. For instance, the W and HSI methods uses the sets $W = \{a, bb\}$ and $H_2 = \{a, b\}$, respectively, to distinguish s_2 in all cases. When sequences to distinguish states

Tab. 17: Sequences to distinguish state s_2.

Sequence	s_2	s_0	s_1	s_3	s_4
a	1	0	0	1	1
b	1	0	1	0	0
aa	10	01	10	11	11
ab	11	00	00	10	11
ba	11	01	11	00	01
bb	10	00	10	00	01
aaa	101	011	011	111	110
aab	100	011	010	111	111
aba	111	001	000	101	111
abb	110	001	000	101	110
baa	111	011	111	001	010
bab	111	010	110	000	011
bba	101	000	100	001	011
bbb	101	000	100	000	010

can be chosen on-the-fly, different strategies can be used to select the sequences. A strategy that is based on cost is presented as follows.

Given a prefix-closed test suite T, we want to know the cost of adding a new test sequence α in T. Algorithm 2 presents a procedure to calculate the cost (discussed in [34]).

Algorithm 2 Calculate Cost.

```
input: test suite T, sequence α
output: integer representing the cost of adding α

if α ∈ T then
  return 0    //case 1
else
  Select the longest sequence β ∈ T such that α = βγ
  if ∄κ ∈ T such that β < κ and β ≠ ϵ then
    return |γ|   //case 2
  else
    return |α| + 1 //case 3
  endif
endif
```

Given a test suite $T = \{\epsilon, b, ba, bb, bbb\}$, we build the testing tree presented in Figure 35. For the test suite T, the tree is composed of white circles and filled edges.

The gray circles and dotted edges represents examples of sequences that are added in T. According to Algorithm 2, the inclusion of these sequences are described as follows.

1. Add the sequence $\alpha = bb$. As $\alpha \in T$ and no change is made in T, the cost is 0 (case 1).

2. Add the sequence $\alpha = ab$. As α do not extend any sequence in T, the cost is its length plus the reset symbol ($|\alpha| + 1 = 3$). (case 3).

3. Add the sequence $\alpha = bba$. Although $bb \in T$, the a involves the creation of a new branch and so its cost is $|\alpha| + 1 = 4$. (case 3).

4. Add the sequence $\alpha = bbba$. The part $bbb \in T$ and is a proper prefix. Thus, this sequence can be extended with a cost $|a| = 1$ (case 2).

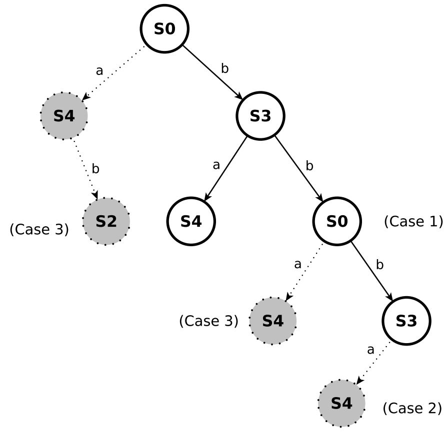


Fig. 35: Testing tree for T.

9.2 Mutants Information

This section provides information about the number of mutants, considering number of states, inputs, outputs, and completeness degree. The number of equivalent mutants is also considered. These mutants have the same configuration used in Section 5.3. We present the data about the mutants as suplementary material since it is not the main focus of this work.

9.2.1 Number of States

Figure 36 shows the average number of mutants in function of the number of states. The total number of mutants clearly grows when the number of states is increased. The growth is quadratic and can be approximated by function $f(n) = 864.6n^2 - 11651.4n + 45653.8$, such that n is the number of states.

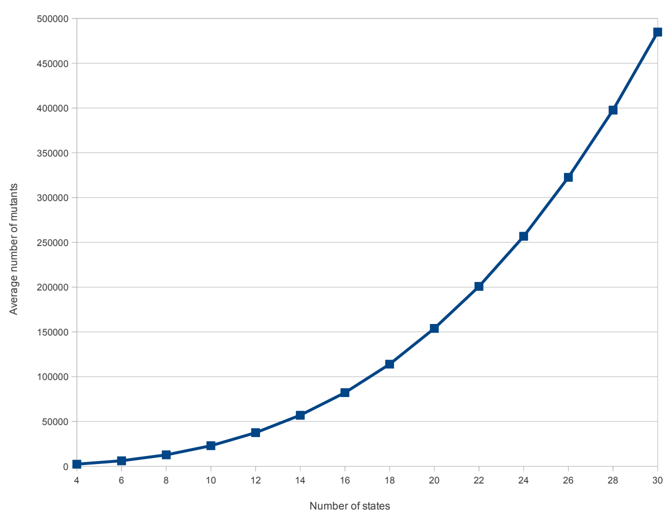


Fig. 36: Number of mutants varying the number of states.

Figure 37 shows the average number of equivalent mutants in function of the number of states. Observe that the number of equivalent mutants grows linearly and can be approximated by function $f(n) = 19.7034n + 18.7785$.

Figure 38 shows the average percentage ($\#em/\#tm$) of equivalent mutants in function of the

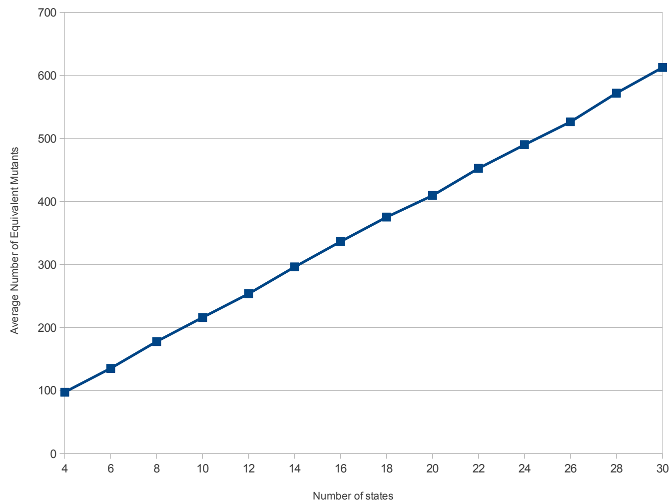


Fig. 37: Number of equivalent mutants varying the number of states.

number of states. As the number of equivalent mutants grows in a small ratio compared with the total number of mutants, the percentage of equivalent mutants clearly decreases with the growth of the number of states. This behavior can be approximated by function $f(n) = 0.494577/(0.38362n^2 + 1.46055n - 0.458584)$.

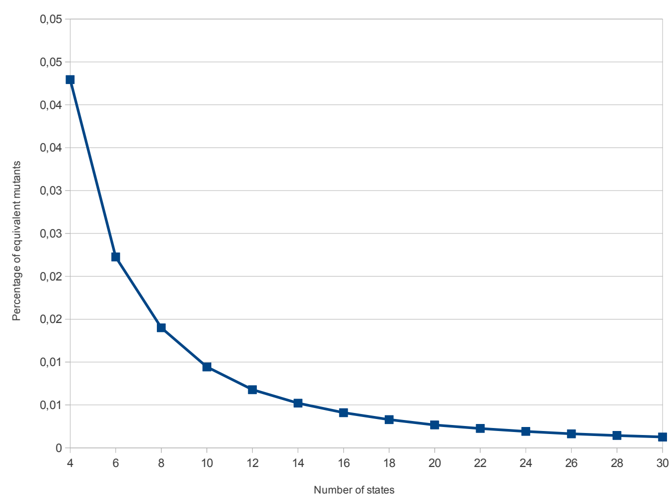


Fig. 38: Percentage of equivalent mutants varying the number of states.

9.2.2 Number of Inputs

Figure 39 shows the average number of mutants in function of the number of inputs. It can also be approximated by a quadratic function $f(i) = 1401.19i^2 + 562.497i - 1806.65$, such that i is the number of inputs.

Figure 40 shows the average number of equivalent mutants in function of the number of inputs. The number of equivalent mutants also presents a growth that can be approximated by the quadratic function $f(i) = 10.6225i^2 + 16.4401i - 21.399$.

Figure 41 shows the average percentage of equivalent mutants in function of the number of inputs. It also shows a tendency of reducing the number of equivalent mutants when increasing the

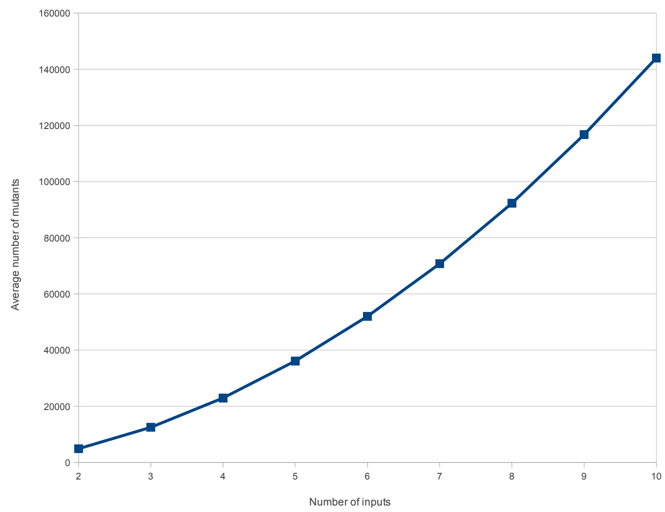


Fig. 39: Number of mutants varying the number of inputs.

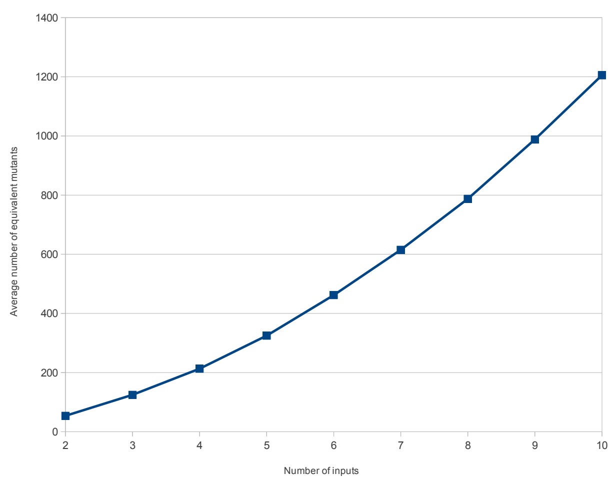


Fig. 40: Number of equivalent mutants varying the number of inputs.

number of inputs.

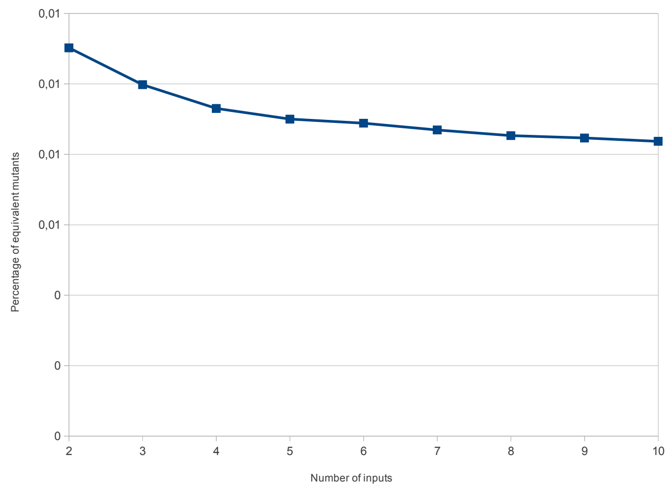


Fig. 41: Percentage of equivalent mutants varying the number of inputs.

9.2.3 Number of Outputs

Figure 42 shows the average number of mutants in function of the number of outputs. Different from states and inputs, it can be approximated by a quadratic function $f(o) = 1729.8o + 15994$, such that o is the number of outputs.

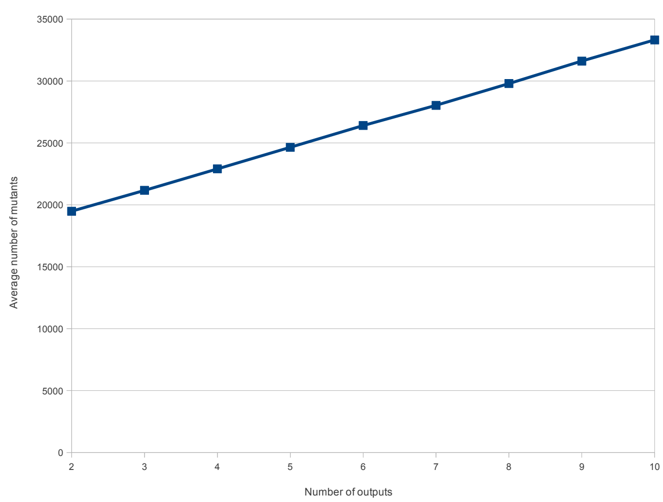


Fig. 42: Number of mutants varying the number of outputs.

Figure 43 shows the average number of equivalent mutants in function of the number of outputs. Although it varies in a small interval $[212, 217]$, it is not possible its behavior.

Figure 44 shows the average percentage of equivalent mutants in function of the number of outputs. This behavior can be approximated by function $f(o) = -0.00590287/(-0.000116558o^2 - 0.0455656o - 0.447827)$.

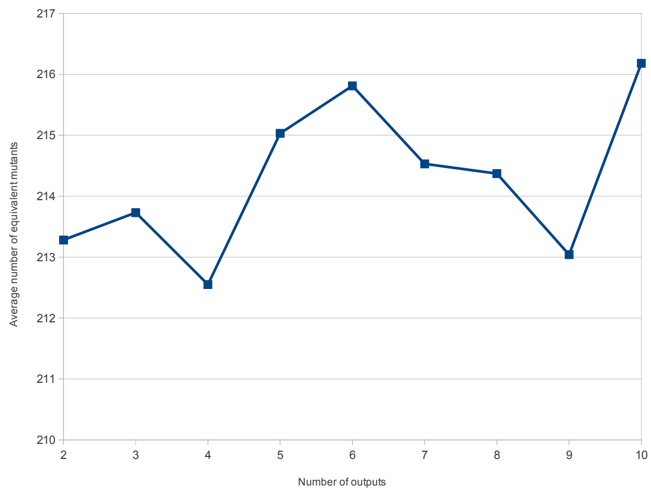


Fig. 43: Number of equivalent mutants varying the number of outputs.

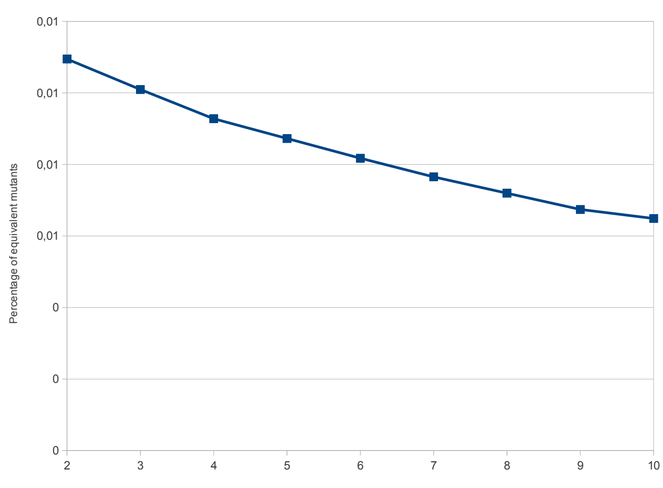


Fig. 44: Percentage of equivalent mutants varying the number of outputs.

9.2.4 Completeness Degree

Figure 45 shows the average number of mutants in function of the number of transitions (completeness degree). This behavior can be approximated by function $f(t) = 22741.1t^2 + 2005.15t - 1688.1)$, such that t is the completeness degree.

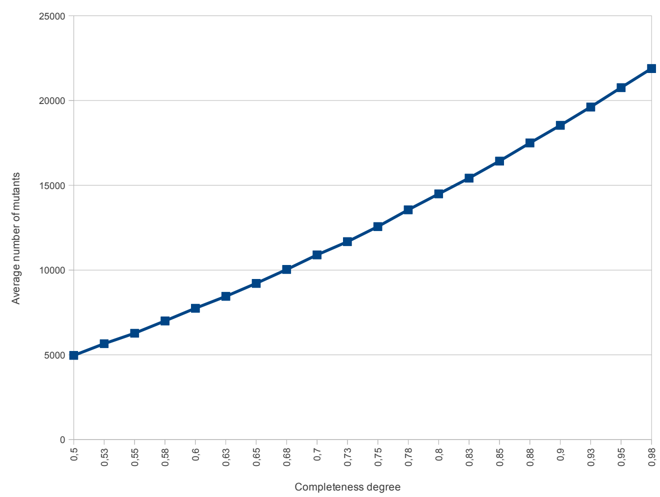


Fig. 45: Number of mutants varying the completeness degree.

Figure 46 shows the average number of equivalent mutants in function of the number of transitions (completeness degree). This behavior can be approximated by function $f(t) = 160.054t^{\mathbf{2}} + 73.041t - 26.2116$.

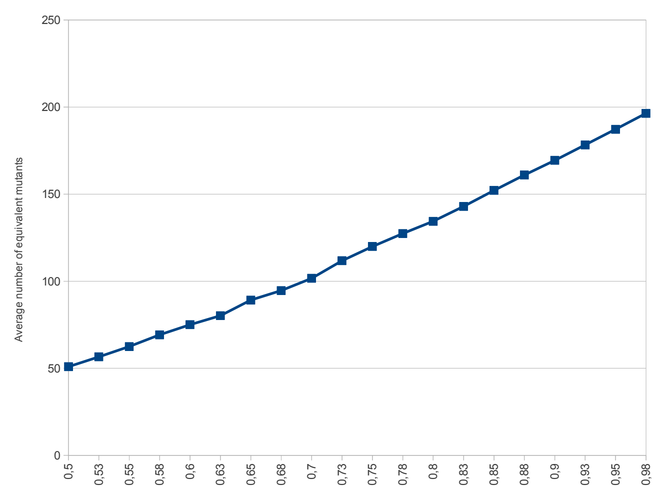


Fig. 46: Number of equivalent mutants varying the completeness degree.

Figure 47 shows the average percentage of equivalent mutants in function of the number of transitions (completeness degree). There is a reduction in the percentage of equivalent mutants when increasing the completeness degree.

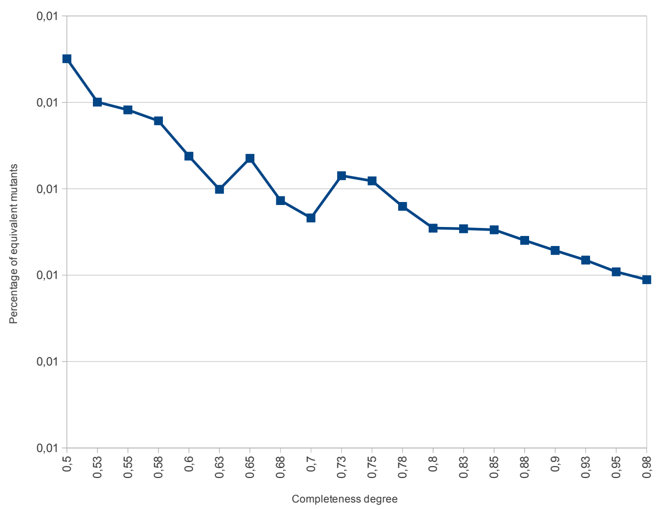


Fig. 47: Percentage of equivalent mutants varying the completeness degree.