



Deposited via The University of Sheffield.

White Rose Research Online URL for this paper:

<https://eprints.whiterose.ac.uk/id/eprint/173162/>

Version: Accepted Version

---

**Article:**

Turker, U.C., Hierons, R. and Jourdan, G.-V. (2021) Minimizing characterizing sets. Science of Computer Programming, 208. 102645. ISSN: 0167-6423

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

---

© 2021 Elsevier. This is an author produced version of a paper subsequently published in Science of Computer Programming. Uploaded in accordance with the publisher's self-archiving policy. Article available under the terms of the CC-BY-NC-ND licence (<https://creativecommons.org/licenses/by-nc-nd/4.0/>).

**Reuse**

This article is distributed under the terms of the Creative Commons Attribution-NonCommercial-NoDerivs (CC BY-NC-ND) licence. This licence only allows you to download this work and share it with others as long as you credit the authors, but you can't change the article in any way or use it commercially. More information and the full terms of the licence here: <https://creativecommons.org/licenses/>

**Takedown**

If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing [eprints@whiterose.ac.uk](mailto:eprints@whiterose.ac.uk) including the URL of the record and the reason for the withdrawal request.

# Minimizing Characterizing sets

Uraz Cengiz Türker<sup>a,\*</sup>, Robert M. Hierons<sup>b</sup>, Guy-Vincent Jourdan<sup>c</sup>

<sup>a</sup>*University of Leicester, School of Informatics, Leicester, UK*

<sup>b</sup>*The University of Sheffield, Department of Computer Science, Sheffield, UK*

<sup>c</sup>*School of Electrical Engineering and Computer Science, Faculty of Engineering, University of Ottawa, Ottawa, CA*

---

## Abstract

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

---

\*Corresponding author

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

1  
2  
3  
4  
5  
6  
7  
8  
9 *Keywords:* Model-based testing, Characterizing set, Complexity

---

## 10 11 **1. Introduction**

12  
13  
14 Testing is an indispensable aspect of a development-cycle for any kind of  
15 system. However, manual testing is typically expensive and error prone. This  
16 has led to significant interest in test automation and, in particular, Model Based  
17 Testing (MBT). MBT techniques and tools use behavioural models and usually  
18  
19 5 Test (MBT). MBT techniques and tools use behavioural models and usually  
20 operate on either a finite state machine (FSM), extended finite state machine  
21 (EFSM) or labelled transition system (LTS) that defines the semantics of the  
22 model. This paper concentrates on testing from a (deterministic) FSM. There  
23 has been significant interest in automating testing based on an FSM model  
24  
25 in areas such as sequential circuits [1], lexical analysis [2], software design [3],  
26  
27 10 in areas such as sequential circuits [1], lexical analysis [2], software design [3],  
28 communication protocols [3, 4, 5, 6, 7, 8, 9, 10], object-oriented systems [11],  
29 and web services [12, 13, 14, 15]. Such techniques have also been shown to be  
30 effective when used in large industrial projects [16].  
31  
32

33 The literature contains many formal methods to automatically generate test  
34  
35 15 suites (sets of test sequences) from FSM models (*specifications*) of systems [9,  
36 17, 18, 19, 20, 21, 22, 23, 24]. These methods represent testing as running  
37 *fault detection experiments* [25]. Formal methods for generating fault detection  
38 experiments are based on particular types of sequences that are derived from the  
39 specification  $M$ . Different test techniques use different types of sequence but we  
40  
41 focus on the use of a *characterizing set* ( $W$ -set), to identify the current state of  
42  
43 20 the implementation, motivated by the fact that every (minimal, deterministic)  
44 FSM has a  $W$ -set. A  $W$ -set is a set of input sequences such that for any pair  
45 ( $s, s'$ ) of distinct states of  $M$  there exists an input sequence  $\bar{x}$  in the  $W$ -set that  
46 separates (distinguishes)  $s$  and  $s'$  [26].  
47  
48  
49  
50

### 51 25 *1.1. Motivation and Problem Statement*

52  
53 Characterizing sets are widely used in test generation. For example, they  
54 are used in the classical *W-method* [3, 27], which has two steps: state recogni-  
55 tion and transition verification. Importantly, test generation involves separately  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

1  
2  
3  
4  
5  
6  
7  
8  
9 following a particular set of input sequences by *every* input sequence from the  
10 characterizing set  $\mathcal{W}$  used. In testing, the system under test is reset before an  
11 30 input sequence is applied, ensuring that the input sequences are applied in the  
12 same state of the system under test.  
13  
14

15 Characterizing sets are used in some other test generation techniques. For  
16 example, the well known HSI-method [28]<sup>1</sup> requires *harmonized state identifiers*  
17 (HSIs) to construct test sequences from a given possibly non-deterministic FSM  
18 35 ( $M$ ). Construction of HSIs requires one to harmonize the elements of a W-set  
19 of  $M$ . That is, in order to construct HSIs for  $M$ , one first has to construct a  
20 W-set [28].  
21  
22  
23  
24

25 A further example, of the use of characterizing sets, can be found in auto-  
26 mated model learning [29]. Here, the W-method is used to decide whether a  
27 40 given hypothesis (FSM) is correct or not [30]. Since testing whether the hypoth-  
28 esis holds (the oracle problem) is computationally expensive it has been referred  
29 to as the bottleneck for learning models from complex systems [31]. Therefore,  
30 any method that generates compact W-sets could help such learning methods  
31 to scale to larger problems.  
32  
33  
34 45  
35

36 It is clear that the size of the test suite returned by a test generation  
37 algorithm that use a characterizing set  $\mathcal{W}$  is directly affected by the num-  
38 ber of input sequences that are in  $\mathcal{W}$ . As explained above, if the W-set  $\mathcal{W}$   
39 contains  $k$  input sequences then test generation techniques such as the W-  
40 method will (separately) follow a set of input sequences by all  $k$  input se-  
41 50 quences in  $\mathcal{W}$ . In addition, there are cases where the reset between test se-  
42 quences is particularly time consuming or expensive since, for example, it may  
43 require a system to be reconfigured or may require manual intervention. As  
44 a result, there has been interest in constructing test generation techniques  
45 55 that return test suites with the minimum number of resets (and so test se-  
46  
47  
48  
49  
50  
51  
52

---

53 <sup>1</sup>Note that in [28] there are two algorithms: the HSI-method for constructing the test suite,  
54 and the HSI-algorithm for constructing state identifiers. Therefore we use *HSI-method* and  
55 *HSI-algorithm* to denote the method and the algorithm respectively.  
56  
57  
58

1  
2  
3  
4  
5  
6  
7  
8  
9 quences) [3, 9, 21, 22, 32, 27, 33, 34, 35, 36, 28, 37, 38, 39].

10 This paper revisits the long-standing problem of finding a smallest charac-  
11 terizing set for an FSM [26]. The aim is to reduce the number of test sequences,  
12 within a test suite, by minimizing the number of elements in W-sets. At a  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000

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

In naming the problems, we differentiate between the *size* of a W-set  $\mathcal{W}$  (the number of input sequences in set  $\mathcal{W}$ ) and the *length* of a W-set  $\mathcal{W}$  (the sum

1  
2  
3  
4  
5  
6  
7  
8  
9 of the lengths of the input sequences in  $\mathcal{W}$ ). As explained above, the size and  
10 length of a  $\mathcal{W}$ -set are both of interest.  
11

### 12 *1.2. Results*

13  
14 We show that the MinSize  $\mathcal{W}$ -set problem is PSPACE-complete. We also  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

90 prove that the MinLength  $\mathcal{W}$ -set problem is NP-complete.

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

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

### 106 *1.3. Practical Implications of our Results and Future Directions*

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

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

#### 1.4. Structure of the paper

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

## 2. Preliminaries

### 2.1. Finite State Machines (FSMs)

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

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

An FSM  $M$  can be represented by a directed graph in which nodes represent states of  $M$  and edges represent transitions of  $M$ . Figure 1 gives an example

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

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

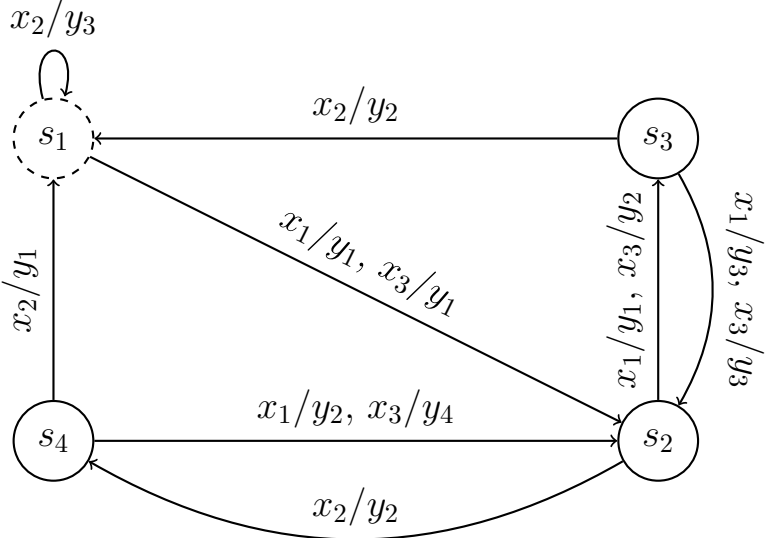


Figure 1: FSM  $M_1$ . Note that the initial state is highlighted with a dashed line.

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

state. FSM  $M$  is said to be *deterministic* if for every state  $s$  and input  $x$ ,  $M$  has only one transition  $\tau = (s, x, y, s') \in h$  with input  $x$  and starting state  $s$ . For example,  $M_1$  is deterministic. In this paper we only consider deterministic FSMs. Although not all FSMs are deterministic, deterministic FSMs have been the main focus of FSM-based test generation.

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

## 2.2. FSM behaviour

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

States  $s, s'$  of  $M$  are *equivalent* if  $L_M(s) = L_M(s')$  and FSMs  $M$  and  $N$  are *equivalent* if  $L(M) = L(N)$ . FSM  $M$  is *minimal* if no two of its states are equivalent. As usual, in this paper we only consider minimal FSMs. This is

1  
2  
3  
4  
5  
6  
7  
8  
9 not a restriction since an FSM can be rewritten to an equivalent minimal FSM  
10 in polynomial time using any technique that minimizes a deterministic finite  
11 automaton. To summarise, we consider deterministic FSMs that are minimal  
12 190 and completely-specified.  
13  
14

15 We also make the usual assumption, when using techniques such as the W-  
16 method, that the system under test has a reliable reset: some process that is  
17 guaranteed to return the system under test to its initial state. A reliable reset  
18 is required in order to ensure that the test sequences are all applied in the same  
19 195 state of the system under test.  
20  
21  
22

### 23 2.3. Characterizing sets

24 The notion of a characterizing set is typically defined in terms of what it  
25 means for an input sequence  $\bar{x}$  to separate two states of an FSM  $M$ .  
26  
27

28 **Definition 1.** *Input sequence  $\bar{x}$  separates states  $s$  and  $s'$  of FSM  $M$  if and only*  
29 200 *if  $M(s, \bar{x}) \cap M(s', \bar{x}) = \emptyset$ .*  
30  
31

32 We now define *W-Sets*.  
33

34 **Definition 2.** *Given FSM  $M$ ,  $W \subseteq X^*$  is a W-Set for  $M$  if for any pair of*  
35 *distinct states of  $M$  there exists a sequence  $\bar{x} \in W$  such that  $\bar{x}$  separates the pair,*  
36 *i.e.,  $\forall s, s' \in S$ , such that  $s \neq s'$ , there is some  $\bar{x} \in W$  with  $M(s, \bar{x}) \cap M(s', \bar{x}) =$*   
37 205  *$\emptyset$ .*  
38  
39  
40

41 Consider again the FSM  $M_1$  in Figure 1. If we consider the set  $\{x_1, x_2\}$   
42 of inputs, the corresponding output responses for the states of  $M_1$  are shown  
43 in Table 1. From this, one can see that every pair of distinct states of  $M_1$  is  
44 separated by either  $x_1$  or  $x_2$  (or both). As a result,  $\mathcal{W} = \{x_1, x_2\}$  is a W-set of  
45 210  $M_1$ .  
46  
47  
48

49 In the next section, we formalise and then explore the complexity of the  
50 following problem.  
51  
52

53 **Definition 3.** *Let  $M$  be a minimal deterministic completely specified FSM, and*  
54 *let  $K$  be a positive integer. In the MinSize W-set problem we are asked to decide*  
55 215 *whether there exists a W-set  $\mathcal{W}$  for  $M$  such that  $|\mathcal{W}| \leq K$ .*  
56  
57  
58

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

Table 1: Response to  $x_1$  and  $x_2$

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

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

### 3. Complexity results

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

#### 3.1. Complexity of the MinSize W-set problem

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

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

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

1  
2  
3  
4  
5  
6  
7  
8  
9  $(s, s', 0)$ ; one copy for each input sequence that the Turing Machine is to guess.  
10 As the process of guessing an input sequence progresses, the value of  $c$  will be  
11 changed to 1 if  $s$  and  $s'$  have been distinguished by the corresponding input  
12 sequence.  
13  
14

15  
16 240 As mentioned, the Turing Machine will guess  $K$  input sequences  $\bar{x}_0, \bar{x}_1, \dots,$   
17  $\bar{x}_{K-1}$  in an iterative manner, extending all sequences by one input in each iter-  
18 ation. There are  $n(n-1)/2$  pairs of states. Since  $M$  has  $n$  states and  $p$  inputs,  
19 encoding a state identifier requires  $O(\log(n))$  space and encoding an input re-  
20 quires  $O(\log(p))$  space. As a result, the Turing Machine requires polynomial  
21 space to store the tuples and the inputs guessed in the current iteration. At  
22  
23 245 each iteration,  $K$  next inputs are guessed non-deterministically. Each guessed  
24 input  $x$  is used to update the corresponding tuples in the natural way (i.e. if  
25  $(s, x, y, s_1), (s', x, y', s'_1) \in h$  then  $(s, s', c_b) \rightarrow (s_1, s'_1, c)$  where  $c = 1$  if  $c_b = 1$  or  
26  $y \neq y'$ , and  $c = 0$  otherwise).  
27  
28  
29  
30

31  
32 250 Consider now how the Turing Machine terminates. First, it terminates with  
33 success if each pair  $(s, s')$  of distinct states has been distinguished;  $s$  and  $s'$  have  
34 been distinguished if for some guessed input sequence the tuple  $(s, s', 0)$  has been  
35 transformed into some  $(s_1, s'_1, 1)$ . The Turing Machine also has a counter  $ctr$ ,  
36 which is increased in each iteration, in order to ensure termination and we now  
37  
38 255 give a bound that can be placed on this counter. If we consider the process  
39 of extending one of the input sequences  $\bar{x}_i$  being guessed, in each iteration we  
40 obtain a sequence of tuples of the form  $(s, s', c)$ ; one for each  $(s, s', 0)$  that  
41 we started with. Clearly, when looking for a bound on sequence length, it is  
42 sufficient to consider sequences  $\bar{x}_i$  such that for every pair of prefixes  $\bar{x}'_i$  and  
43  $\bar{x}''_i$  of  $\bar{x}_i$ , with  $|\bar{x}'_i| < |\bar{x}''_i|$ , we have that  $\bar{x}'_i$  and  $\bar{x}''_i$  define different sequences of  
44 tuples (otherwise, we can replace  $\bar{x}''_i$  by  $\bar{x}'_i$  to obtain a shorter sequence). There  
45 are  $2n^2$  possible values for each tuple and  $n(n-1)/2$  tuples and so at most  
46  $2^{n(n-1)/2} n^{n(n-1)}$  possible values for the tuples. Thus, the Turing Machine can  
47  
48 260 terminate with failure if  $c$  exceeds this bound. As a result, the space required to  
49 store the counter is of  $O(\log_2(2^{n(n-1)/2} n^{n(n-1)}))$  and so only polynomial space  
50  
51  
52  
53  
54  
55 265 is required. The result therefore follows.  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

1  
2  
3  
4  
5  
6  
7  
8  
9 We now show that the MinSize W-set problem is PSPACE-complete.

10  
11 **Theorem 1.** *The MinSize W-set problem for completely specified deterministic*  
12 *FSMs is PSPACE-complete.*  
13  
14

15 270 PROOF. First, by Proposition 1 we know that the problem is in PSPACE. We  
16 now show that the problem is PSPACE-hard. Consider the case where  $K = 1$ .  
17 Then there is a W-set of size at most  $K$  for  $M$  if and only if there is a preset  
18 distinguishing sequence (PDS)<sup>2</sup> for  $M$ . Thus, any algorithm that can decide  
19 whether an FSM has a W-set of size  $K$  can also be used to decide whether an  
20 FSM has a PDS. Since the problem of deciding whether a deterministic FSM  
21 has a PDS is PSPACE-hard [41], we have that the problem of deciding whether  
22 an FSM has a W-set of size at most  $K$  is also PSPACE-hard. The result thus  
23 275 follows.  
24  
25  
26  
27  
28  
29

### 30 3.2. Complexity of the MinLength W-set problem

31 280 The motivation for the work in this paper is that a small W-set is likely  
32 to lead to a small test suite and here we are interested in how many input  
33 sequences are contained in the W-set. However, as discussed earlier, we might  
34 instead be interested in the overall length of a W-set  $\mathcal{W}$ : the sum of the lengths  
35 of the input sequences in  $\mathcal{W}$ . We now consider the complexity of this problem  
36 for deterministic FSMs.  
37  
38  
39 285  
40

41 We will first show that the problem is NP-hard by relating it to the Hitting  
42 Set Problem.  
43  
44

45 **Definition 5.** *Let us suppose that  $A$  is a finite set,  $\{A_1, \dots, A_k\}$  is a set of*  
46 *subsets of  $A$ , and  $K$  is an integer. The Hitting Set Problem is to decide whether*  
47 *there exists some subset  $A'$  of  $A$ , of size at most  $K$ , such that every  $A_i$  contains*  
48 290 *at least one element of  $A'$  ( $A_i \cap A' \neq \emptyset$ ).*  
49  
50  
51

52 The hitting set problem is known to be NP-complete [42]. We use this result  
53 to prove that the MinLength W-set problem is NP-hard.  
54  
55

---

56 <sup>2</sup>An input sequence  $\bar{x}$  is a PDS for  $M$  if  $\bar{x}$  distinguishes all of the states of  $M$ .  
57  
58

1  
2  
3  
4  
5  
6  
7  
8  
9 **Proposition 2.** *The MinLength W-set problem is NP-hard.*

10  
11 295 PROOF. Let us assume that we are given an instance of the Hitting Set Problem  
12 defined by set  $A = \{a_1, \dots, a_p\}$ , set  $\mathcal{A} = \{A_1, \dots, A_n\}$  of subsets of  $A$ , and  
13 integer  $K$ . We will show how one can construct an FSM  $M_{A,\mathcal{A}}$  such that  $M_{A,\mathcal{A}}$   
14 has a W-set of length at most  $K$  if and only if there is a solution to the instance  
15 of the Hitting Set Problem defined by  $A$ ,  $\mathcal{A}$ , and  $K$ .  
16  
17

18  
19 300 The FSM  $M_{A,\mathcal{A}}$  will have one state  $s_j$  for each element  $A_j$  of  $\mathcal{A}$  ( $1 \leq j \leq n$ ).  
20 There is an additional state  $s_0$ , which is also the initial state. We will have  
21 output set  $Y$  that, for all  $1 \leq i \leq n$  and  $1 \leq j \leq p$ , contains a unique output  
22  $y_{ij}$ , as well as another output  $y_0$ . Thus,  $Y$  contains  $np + 1$  outputs. For each  
23  $a_i \in A$  we will introduce a unique input  $x_i$  and we will add transitions so that  
24  $x_i$  distinguishes state  $s_j$  from other states (including  $s_0$ ) if and only if  $a_i \in A_j$ .  
25 305 This will be achieved through the following transitions ( $1 \leq i \leq n, 1 \leq j \leq p$ ).  
26  
27  
28  
29

- 30  
31 1. If  $a_i \in A_j$  then  $M_{A,\mathcal{A}}$  has transition  $(s_j, x_i, y_{ij}, s_j)$ .  
32 2. If  $a_i \notin A_j$  then  $M_{A,\mathcal{A}}$  has transition  $(s_j, x_i, y_0, s_j)$ .  
33

34 For all  $1 \leq i \leq p$ ,  $M_{A,\mathcal{A}}$  also has the transition  $(s_0, x_i, y_0, s_0)$ .  
35

36 310 Note that  $M_{A,\mathcal{A}}$  is not initially connected as all its transitions loop. It  
37 is straightforward to extend this FSM to form an initially connected FSM by  
38 introducing additional input symbols and transitions that connect the initial  
39 state with all other states but that cannot be used for distinguishing the states  
40 of  $M_{A,\mathcal{A}}$ . Specifically, for all  $1 \leq j \leq n$  we introduce a new input  $x'_j$  that reaches  
41 state  $s_j$  from the initial state. The transitions with input  $x'_j$  are defined by: for  
42 315 every state  $s_k, 0 \leq k \leq n$ , the transition from  $s_k$  with input  $x'_j$  takes the FSM  
43 to state  $s_j$  with output  $y_0$ .  
44  
45  
46  
47

48 Let us suppose that  $X'$  is a subset of  $X$ . By construction, we have that  $X'$   
49 distinguishes  $s_j$  from the other states of  $M_{A,\mathcal{A}}$  if and only if there exists  $x_i \in X'$   
50 such that  $a_i \in A_j$ . Thus,  $X'$  is a W-set for  $M_{A,\mathcal{A}}$  if and only if  $\{a_i | x_i \in X'\}$  is  
51 320 a hitting set.  
52  
53  
54

55 We now know that any algorithm that solves the MinLength W-set problem  
56 for  $M_{A,\mathcal{A}}$  and  $K$  has also solved the Hitting Set Problem defined by  $A$ ,  $\mathcal{A}$ ,  
57  
58

1  
2  
3  
4  
5  
6  
7  
8  
9 and  $K$ . In addition,  $M_{A,\mathcal{A}}$  can be constructed in polynomial time. The result  
10 therefore follows from the Hitting Set Problem being NP-hard.  
11

12  
13 The problem is also in NP. The following result is easier to prove because  
14 we know that an FSM has a characterising set of length at most  $(n - 1)^2$ .  
15

16  
17 **Proposition 3.** *The MinLength W-set problem is in NP.*  
18

19  
20 PROOF. First observe that a deterministic FSM with  $n$  states has a W-set if  
21 and only if it has a W-set with no more than  $n - 1$  input sequences, each of  
22 length at most  $n - 1$ . Thus, we can place an upper bound of  $(n - 1)^2$  on the  
23 value of  $K$  considered.  
24

25  
26 A non-deterministic Turing Machine can simply guess a set  $\mathcal{W} = \{\bar{x}_1, \dots, \bar{x}_k\}$   
27 of input sequences of total length at most  $K$ . It is then sufficient to check  
28 whether this set  $\mathcal{W}$  of input sequences is a W-set. For every pair  $s, s'$  of distinct  
29 states of  $M$ , the algorithm proceeds through up to  $k$  iterations. Here, in the  
30  $i$ th iteration we compute  $M(s, \bar{x}_i)$  and  $M(s', \bar{x}_i)$ . If  $M(s, \bar{x}_i) \neq M(s', \bar{x}_i)$  then  
31 the Turing Machine records that  $s$  and  $s'$  are distinguished by  $\mathcal{W}$  and moves  
32 on to check the next pair of states. If all pairs of states have been considered,  
33 and found to be distinguished by  $\mathcal{W}$  then the Turing Machine terminates with  
34 success. If  $M(s, \bar{x}_i) = M(s', \bar{x}_i)$  and  $i < k$  then we move to iteration  $i + 1$  and  
35 otherwise ( $M(s, \bar{x}_i) = M(s', \bar{x}_i)$  and  $i = k$ ) the Turing Machine concludes that  
36  $\mathcal{W}$  is not a W-set.  
37

38  
39 The outer loop of this algorithm iterates at most  $n(n - 1)$  times. In addition,  
40 the inner loop iterates at most  $|\mathcal{W}| \leq (n - 1)^2$ . The number of iterations is thus  
41 of  $O(n^4)$  times. Further, given input sequence  $\bar{x}_i$  and state  $s$  it is possible to  
42 compute  $M(s, \bar{x}_i)$  in polynomial time.  
43

44  
45 To conclude, this algorithm takes polynomial time and returns success if  
46 and only if  $\mathcal{W}$  is a W-set (it succeeds in separating all pairs of states). Thus,  
47 a non-deterministic Turing Machine can solve the problem in polynomial time  
48 and so the result follows.  
49

50  
51 From the above, we have the following result.  
52  
53  
54  
55

1  
2  
3  
4  
5  
6  
7  
8  
9 **Theorem 2.** *The MinLength W-set problem is NP-complete.*

10  
11  
12 **4. Algorithm for Constructing W-sets.**

13  
14  
15 355 This section introduces a novel algorithm that takes a deterministic minimal  
16 FSM and tries to generate a W-set with relatively few elements (please see  
17 Algorithm 1 for details). It does so without specifying a bound ( $K$ ) on the  
18 number of elements.  
19

20  
21 The algorithm receives an FSM  $M$  and performs a breadth-first search  
22 (BFS), on input sequences, and checks for solutions using a greedy heuristic.  
23 360 The BFS continues until one of the following termination conditions holds.  
24

- 25  
26 1. A W-set is formed; or  
27  
28 2. An upper bound on the depth of the tree is exceeded. We set this bound as  
29  $n - 1$  since, for an FSM with  $n$  states, every pair of states can be separated  
30 by an input sequence of length at most  $n - 1$  [26].  
31 365

32  
33 Upon receiving its input, the algorithm initiates a *pair set*  $\Delta$  (Line 1 of  
34 Algorithm 1). The pair set ( $\Delta$ ) initially contains the set of all pairs of distinct  
35 states (i.e., the set defined by  $\binom{S}{2}$ ) and is used to check which pairs have been  
36 separated so far. The algorithm uses  $\Delta$  to keep the remaining pairs to be  
37 separated and hence the algorithm terminates with success if  $|\Delta| = \emptyset$ . The  
38 370 algorithm applies the BFS iteratively, constructing a tree structure that we call  
39 a *BFS tree*. A BFS tree contains a set of vertices  $V$ , where each vertex  $v \in V$  has  
40 four pieces of information: a set of *current states*  $v_C$ ; a set of *initial states*  $v_I$ ; an  
41 input sequence  $v(\bar{x})$ ; and finally an output sequence  $v(\bar{y})$ . These are related by:  
42  
43 375 for all  $s \in v_I$ , there exists a corresponding  $s' \in v_C$  such that  $(s, \bar{x}, \bar{y}, s') \in h^*$ .  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

---

**Algorithm 1:** Minimal W-set for  $M$

---

**Input:** FSM  $M = (S, s_0, X, Y, h)$  such that  $|S| > 1$   
**Output:** A W-set for  $M$

**begin**

```

1   $\Delta \leftarrow \{(s_i, s_j) \mid s_i, s_j \in S \text{ and } i < j\}$ 
2   $v \leftarrow (S, S, \varepsilon, \varepsilon), \text{push}(v, V)$ 
3   $\ell \leftarrow 0.$ 
4  while  $\ell \leq n - 1$  do
5       $V^* \leftarrow \emptyset$ 
6      while  $V \neq \emptyset$  do
7           $v \leftarrow \text{pop}(V)$ 
8          foreach input symbols  $x \in X$  do
9              Retrieve  $P(v_C, x).$ 
10             foreach  $v^y \in P(v_C, x)$  do
11                  $v_C^y = \{s' \in S \mid \exists s \in v_C.(s, x, y, s') \in h\}$ 
12                  $v_I^y = \{s \in v_I \mid \bar{x}/\bar{y} \in L_M(s)\}$ 
13                  $v^y(\bar{x}) = v(\bar{x})x, v^y(\bar{y}) = v(\bar{y})y$ 
14                  $\text{push}(v^y, V^*)$ 
15          $V \leftarrow V^*$ 
16          $\ell \leftarrow \ell + 1$ 
17          $Seq \leftarrow \langle \rangle$ 
18         foreach  $v \in V$  do
19              $\chi \leftarrow$  number of pairs removed from set  $\Delta$  by  $v(\bar{x})$ 
20              $Seq \leftarrow Seq \langle (\chi, v(\bar{x})) \rangle$ 
21          $\text{Sort}(Seq)$ 
22         foreach  $\chi \in Seq$  do
23             Remove pairs separated by  $\bar{x}^*$  from  $\Delta$ 
24             if A pair has been separated then
25                  $\mathcal{W} \leftarrow \mathcal{W} \cup \{\bar{x}^*\}$ 
26         if  $\Delta = \emptyset$  then
27             Return  $\mathcal{W}$  after prefix removal.

```

---

1  
2  
3  
4  
5  
6  
7  
8  
9 The BFS tree is constructed as follows. In each iteration (considering a  
10 given depth), the algorithm processes the set  $V$  that contains the vertices that  
11 correspond to the current leaves of the tree. Initially, the vertex set  $V$  has a  
12 single element  $v$  such that  $v_C = S$ ,  $v_I = S$ , and  $v(\bar{x}) = v(\bar{y}) = \varepsilon$  (Line 2  
13 of Algorithm 1). In each iteration, the algorithm, while processing a vertex  
14  $v \in V$ , then separately *applies* all the inputs from  $x \in X$  to  $v_C$  (Lines 8-14 of  
15 Algorithm 1). In each iteration, the algorithm, while processing a vertex  
16  $v \in V$ , then separately *applies* all the inputs from  $x \in X$  to  $v_C$  (Lines 8-14 of  
17 Algorithm 1). By considering the output produced by  $M$ , in response to  $x$ , the  
18 state sets of  $v$  are *partitioned*; for all  $y$  such that  $x/y \in M(v_C, x)$ , we introduce  
19 a new vertex  $v^y$  such that  $v_C^y = \{s' \in S \mid \exists s \in v_C. (s, x, y, s') \in h\}$ ,  $v^y(\bar{x}) = v(\bar{x})x$ ,  
20  $v^y(\bar{y}) = v(\bar{y})y$ , and we form  $v_I^y$  by simply inheriting the corresponding initial  
21 states of  $s$ . We use  $P(v_C, x)$  to denote the set of vertices that are created by  
22 applying  $x$  to  $v_C$ . The newly created vertices are then added to a set  $V^*$  (Line 14  
23 of Algorithm 1). When all the vertices of the current level have been processed,  
24 the algorithm copies  $V^*$  to  $V$  (Line 15 of Algorithm 1), increments the level  
25 variable ( $\ell$ ) by one (Line 16 of Algorithm 1) and initiates a sequence  $Seq$  (Line  
26 17 of Algorithm 1).  
27  
28  
29  
30  
31  
32  
33

34 This is then followed by analyzing the outcome of the BFS step. In order  
35 to do this, the algorithm first gathers all the distinct input sequences from the  
36 vertices in the set  $V^*$  and forms a set  $\bar{X}$ . Then, for each input sequence  $\bar{x} \in \bar{X}$ ,  
37 it counts the number of pairs of states from  $\Delta$  that are separated by  $\bar{x}$  and  
38 stores this value in  $\chi$  (Lines 18-20 of Algorithm 1). Note that the algorithm  
39 does not remove a pair of states from  $\Delta$  at this step but it removes pairs after  
40 processing all the vertices in  $V$  (see Lines 22-25), it only counts the number of  
41 pairs that can be removed by the input sequence under consideration. Then the  
42 algorithm associates this value with the input sequence, i.e.,  $(\bar{x}, \chi)$ , and stores  
43 this in set  $Seq$ . The algorithm then sorts  $Seq$  according to the  $\chi$  values (Line  
44 21 of Algorithm 1).  
45  
46  
47  
48  
49  
50  
51

52 After this, the algorithm moves to an iterative step in which it removes  
53 pairs (through a heuristic step) from  $\Delta$  ( $\Delta$  stores the pairs of states not yet  
54 separated). The algorithm uses a (greedy) heuristic in which it selects an input  
55 sequence  $(\bar{x})$ , from one of the new vertices, that is associated with one of the  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

1  
2  
3  
4  
5  
6  
7  
8  
9 largest  $\chi$  values. That is, it selects an input sequence that separates as many  
10 states as possible. If the algorithm can eliminate one or more pairs from  $\Delta$  with  
11  $\bar{x}$ , then the algorithm adds  $\bar{x}$  to  $\mathcal{W}$ . Note that  $\bar{x}$  will not always be included;  
12  
13  
14  
410 it may be that the pairs of states separated by  $\bar{x}$  have been separated by input  
15 sequences previously added. This process continues until all pairs have been  
16 separated (or it runs out of input sequences) (Lines 22-25 of Algorithm 1).  
17

18  
19 At the end of an iteration, if  $\Delta$  is empty then the algorithm returns  $\mathcal{W}$  after  
20 removing all the sequences that are proper prefixes of other sequences (i.e. given  
21  
22  
415 sequences  $\mathcal{W} = \{x_1x_2, x_1\}$  we drop  $x_1$  and we have  $\mathcal{W} = \{x_1x_2\}$ ) (Lines 26-27  
23 of Algorithm 1). Otherwise the algorithm continues to execute. We now show  
24 that if Algorithm 1 terminates with success, then the set  $\mathcal{W}$  returned defines a  
25 W-set for  $M$ .  
26  
27

28  
29 **Proposition 4.** *Let us suppose that  $v$  is a vertex formed during the BFS and*  
30  
420 *that  $v^p$  is the parent vertex of  $v$ . If states  $s$  and  $s'$  are both in  $v_I(v^p)$  but only*  
31 *one of  $s$  and  $s'$  is in  $v_I(v)$  then  $v(\bar{x})$  is a separating sequence for  $s$  and  $s'$ .*  
32  
33

34 PROOF. The result follows from the definition of a separating sequence.  
35

36 Therefore, if a pair  $(s, s')$  of states is removed from  $\Delta$  then the algorithm has  
37 computed a separating sequence for  $s$  and  $s'$ . Since the algorithm terminates  
38  
39  
425 when all the pairs have been removed from  $\Delta$ , we have the following result.  
40

41  
42 **Theorem 3.** *Let us suppose that  $M$  is a completely specified minimal deter-*  
43 *ministic FSM. If Algorithm 1, when given  $M$ , returns non-empty set  $\mathcal{W}$ , then*  
44  *$\mathcal{W}$  is a W-set for  $M$ .*  
45  
46

47 In the worst case, the proposed algorithm constructs a BFS tree by applying  
48  
49  
430 every input sequence of length up to  $n - 1$  and so the worst time complexity of  
50 the algorithm is exponential. In the next section we report on experiments that  
51 evaluated this algorithm, finding that (compared to the existing algorithm [3])  
52 the proposed algorithm reduces the number of test sequences and the number  
53 of inputs in the test suite by 37.3% and 36.4% on average respectively and does  
54  
55  
56  
435 so within a reasonable time.  
57  
58

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

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

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

## 5. Empirical Evaluation

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

We used an Intel I7 CPU with 32GB RAM to carry out the experiments. We implemented the W-set construction algorithm (from now on  $EA$ ) as given

1  
2  
3  
4  
5  
6  
7  
8  
9 in [26], the W-method as given in [3] and the proposed algorithm using C++  
10 on Microsoft Visual studio .Net 2013.  
11

### 12 5.1. Existing W-set generation algorithm and the W-method 13

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

15 In the first step, the algorithm determines the set of pairs of states ( $\mathcal{S}$ ) that  
16 can be separated by a single input symbol. While doing this the algorithm  
17 associates a pair of states in  $\mathcal{S}$  with the input sequence (which has only one  
18 input) that separates them.  
19  
20  
21  
22

23 In the second phase, the algorithm enters a loop that ends when all pairs  
24 are distinguished. At each iteration of the loop, the algorithm finds pairs of the  
25 form  $(s, s')$  such that  $s$  and  $s'$  have not yet been separated and there is some  
26 input  $x$  that takes  $(s, s')$  to a pair that has already been separated by some  
27 input sequence  $\bar{x}$ . When the algorithm finds such a pair  $(s, s')$ , it associates  
28  $(s, s')$  with the input sequence  $x\bar{x}$ . The original algorithm given in [26] uses a  
29 set of tables (called  $P_k$  tables). In our implementation, we used lists instead of  
30 tables. Please see Algorithm 2 for more details.  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

---

**Algorithm 2:** The W-Set generation algorithm.

---

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

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

**begin**

```
1   $S \leftarrow \{\}$ 
2  foreach  $(s, s') \in S, s \neq s'$  do
3      if there exist  $x, y, y'$  such that  $x/y \in L_M(s), x/y' \in L_M(s')$  and  $y \neq y'$ 
4          (randomly pick one such  $x \in X$ ) then
5               $S \leftarrow S \cup \{(s, s'), x\}$ 
6
7  while Not all pairs of states of  $M$  are separated do
8       $S' \leftarrow \{\}$ 
9      foreach pair of state  $(s, s')$  that is not included in  $S$  do
10         if there exist  $x, y, \bar{x}, s_1, s'_1$  such that  $(s, x, y, s_1) \in h, (s', x, y, s'_1) \in h$  and
11              $(s_1, s'_1, \bar{x}) \in S$  then
12                  $S' \leftarrow S' \cup \{(s, s'), x\bar{x}\}$ 
13
14      $S \leftarrow S \cup S'$ 
15
16 foreach  $\{(s, s'), \bar{x}\} \in S$  do
17      $\mathcal{W} \leftarrow \mathcal{W} \cup \bar{x}$ 
18
19 Return  $\mathcal{W}$  after prefix removal.
```

---

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

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

475

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

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

480

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

$$TS = r.R.W \cup r.R.X.W$$

where  $r$  denotes the reset operation that brings the FSM to the initial state,

and  $R$  denotes the set of prefix closed sequences that reaches each state from the initial state. The details of the W-method are given in Algorithm 3.

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

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

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

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

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

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

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

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

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

1  
2  
3  
4  
5  
6  
7  
8  
9 Therefore  $TS_1$  contains 9 sequences with 31 inputs.

10 Now suppose that Algorithm 3 is given  $M_1$  and  $\mathcal{W}_2$  as inputs. The algorithm  
11 will construct the same  $R$  and  $Z$  sets. However, because  $\mathcal{W}_2 = \{x_1, x_2\}$ , after  
12 prefix removal, the test suite set  $TS_2$  is as follows.  
13  
14

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

15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25 The test suite  $TS_2$  contains 18 sequences with 62 inputs. So in this example,  
26 the proposed method reduced both the number of inputs and the number of  
27 sequences by 50%.  
28  
29

---

30 **Algorithm 3:** The W-Method.

---

31 **Input:** FSM  $M = (S, s_0, X, Y, h)$ , and W-set  $\mathcal{W}$  for  $M$

32 **Output:** A test suite  $TS$  for  $M$

33 **begin**

```

34 1 Initialize  $R \leftarrow \{\varepsilon\}$ ,  $Z \leftarrow \emptyset$ , and  $TS \leftarrow \emptyset$ 
35 2 foreach  $s \in S \setminus \{s_0\}$  do
36 3   Find an input sequence  $\bar{x}$  that takes  $M$  from  $s_0$  to  $s$  and add this sequence to  $R$ 
37    $(R \cup \{\bar{x}\})$ 
38 4 foreach  $\bar{x} \in R$  do
39 5   foreach  $x \in X \cup \{\varepsilon\}$  do
40 6      $Z \cup \{\bar{x}x\}$ 
41 7 foreach  $\bar{x} \in Z$  do
42 8   foreach  $\bar{x}' \in \mathcal{W}$  do
43 9      $TS \leftarrow TS \cup \{\bar{x}\bar{x}'\}$ 
44 10 Return  $TS$  after prefix removal.

```

---

48  
49  
50  
51 *5.2. Research questions*

52 The experiments were designed to address the following research questions.

53 RQ1 Does the proposed algorithm tend to return W-sets with fewer input se-  
54 quences than the standard W-set generation algorithm?  
55  
56  
57  
58

1  
2  
3  
4  
5  
6  
7  
8  
9 RQ2 Does the proposed algorithm tend to lead to test suites with fewer input  
10 sequences, when the  $W$ -set is used within the  $W$ -method?

11  
12 RQ3 Does the proposed algorithm tend to lead to test suites with fewer inputs  
13 in total (sum of lengths of input sequences), when the  $W$ -set is used within  
14 the  $W$ -method?  
15  
16

17 RQ4 Does the proposed algorithm run in a reasonable amount of time?  
18

19  
20 Observe that the first research question directly relates to the objective of the  
21 proposed algorithm: to return a  $W$ -set with relatively few input sequences. The  
22 second research questions is related to the motivation for producing such a  $W$ -  
23  
24 set, which is that its use should lead to test suites with a relatively small number  
25 of input sequences (and so, also, resets). Thus, positive answers to the first two  
26 research questions would suggest that the algorithm is effective in returning  
27 small  $W$ -sets and its use leads to relatively small test suites. We considered the  
28 third research question in order to check that any benefits of having smaller test  
29  
30 suites (fewer input sequences) are not potentially outweighed by the overall test  
31 suite length (the sum of the lengths of the input sequences). The final research  
32 question addresses the scalability of the proposed algorithm.  
33  
34  
35  
36

### 37 38 *5.3. Experimental subjects*

39  
40 In order to compare the proposed algorithm and the existing  $W$ -set gener-  
41 ation algorithm, we randomly generated two classes of FSMs and we also used  
42 benchmark FSMs. In this section, we provide more details regarding the FSMs  
43 used.  
44  
45  
46

#### 47 *5.3.1. FSMs in CLASS I*

48  
49 The FSMs in the first class ( $C1$ ) were generated as follows. First, for each  
50 input  $x$  and state  $s$  we randomly assigned the next state and output values.  
51 After an FSM  $M$  was generated we checked its suitability as follows. We checked  
52 whether  $M$  was initially connected and was also minimal (and so has a  $W$ -set).  
53 If the FSM passed these tests then we included it into  $C1$ , otherwise we omitted  
54  
55  
56  
57  
58

1  
2  
3  
4  
5  
6  
7  
8  
9 this FSM and produced another one. Consequently, all generated FSMs were  
10 initially connected, minimal, and had W-sets.  
11

12 By using this procedure, we constructed six sets of 1000 FSMs with  $n$  states,  
13 for each  $n \in \{50, 60, \dots, 150\}$ . In all cases, there were three input symbols and  
14 three output symbols. In total, we therefore constructed 11,000 FSMs for the  
15 first class of FSMs.  
16  
17  
18

19  
20 5.3.2. *FSMs in CLASS II*  
21

22 Note that for the FSMs in C1, the next state of each transition is randomly  
23 selected, hence the in-degrees<sup>3</sup> of different states will be similar. In order to  
24 reduce the potential impact of this, we also generated a second class (C2),  
25 where we aimed to have FSMs with less similar in-degree values. To create  
26 such an FSM, we first randomly generated an FSM as before and then selected  
27 a subset  $\bar{S}$  of states, the intention being to give these states higher in-degree  
28 values than the states in  $S \setminus \bar{S}$ . To create higher in-degree values for the states in  
29  $\bar{S}$ , we randomly selected a subset  $\Gamma$  of the set of transitions (where each element  
30 of  $\Gamma$  is a pair  $(s, x)$  denoting the transition from state  $s$  with input symbol  $x$ ).  
31  
32 We then forced the transitions in  $\Gamma$  to end in states in  $\bar{S}$ . Similar to C1, after  
33 an FSM  $M$  was generated we checked its suitability.  
34  
35  
36  
37  
38

39 When constructing this set of FSMs it was necessary to choose the cardinal-  
40 ities of  $\bar{S}$  and  $\Gamma$ . If  $|\Gamma|$  was too large and  $|\bar{S}|$  was too small then one might not  
41 be able to construct a connected FSM, or one might tend to construct FSMs  
42 that are not minimal and so do not have a W-set. On the other hand, if  $|\Gamma|$  was  
43 too small and  $|\bar{S}|$  was too large then the in-degrees of the states will again be  
44 similar.  
45  
46  
47

48 In these experiments we chose  $|\bar{S}|$  to be 10% of the states and we set  $|\Gamma|$  to  
49 be 30% of the transitions. We observed that if  $|\Gamma|$  is a much higher proportion  
50 of the number of transitions then it takes too much time to construct suitable  
51 FSMs.  
52  
53

---

54  
55 <sup>3</sup>The in-degree of a state  $s$  is the number of transitions ending in  $s$ .  
56  
57  
58

1  
2  
3  
4  
5  
6  
7  
8  
9 We constructed 11,000 FSMs in  $C2$ , with the number of states  $n$  being in  
10  $\{50, 60, \dots, 150\}$ ; for each  $n$  there were 1,000 FSMs. We again used input and  
11 output alphabets of size three.  
12

### 13 14 15 570 5.3.3. Benchmark FSMs

16 In addition to the randomly generated FSMs, we used benchmark FSM  
17 specifications from a repository<sup>4</sup>. The FSM specifications were available in the  
18 *Dot* format. In order to process the FSMs, we converted the *Dot* file format to  
19 our FSM specification format. We used 16 FSMs from the benchmark; these  
20 were minimal, deterministic, had a  $W$ -set and had fewer than 10 input bits<sup>5</sup>.  
21 575 We can split the set of benchmark FSMs used into groups. The first group  
22 originated from the ACM/SIGDA benchmarks, a set of test suites (FSMs) used  
23 in workshops between 1989 and 1993 [43] These FSMs, ranging from simple  
24 circuits to circuits obtained from industry, were: *DVRAM*, *Ex4*<sup>6</sup>, *Log*, *Rie*, and  
25 *Shift Register*. For these FSMs we flattened the transitions with don't care ("–  
26 ") symbols. That is, for each string with don't care symbol, we introduce two  
27 strings with 0 and 1 symbols, i.e., –01010 will be replaced to lead to 101010 and  
28 001010 We also used FSMs that model the alternating bit protocol and FSMs  
29 obtained by learning algorithms applied to Coffee Machine and Maestro credit  
30 card case studies [44]. Table 2 gives the number of states and transitions of the  
31 580 benchmark FSMs.  
32  
33  
34  
35  
36  
37  
38  
39 585  
40  
41  
42

### 43 5.4. Measures used

44 The experiments applied the proposed algorithm and the previously pub-  
45 lished algorithm, using these to generate  $W$ -sets and then test suites (using the  
46  $W$ -method). Thus, for each algorithm and group of FSMs, such as all FSMs  
47 590  
48  
49

---

50  
51 <sup>4</sup>The repository can be accessed via <https://automata.cs.ru.nl/>.

52 <sup>5</sup>This was for practical reasons since the circuits receive inputs in bits and  $b$  bits correspond  
53 to  $2^b$  inputs.

54 <sup>6</sup>FSM specification *Ex4* is partially specified. We complete the missing transitions by  
55 adding self looping transitions with a special output symbol, and do not use these inputs for  
56  $W$ -set construction.  
57  
58

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

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

Table 2: Benchmark FSMs and their sizes

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

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

1  
2  
3  
4  
5  
6  
7  
8  
9 W-set returned by  $P$ , then this measure can be defined as follows.

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

10  
11  
12  
13  
14 The second measure ( $M2$ ) gives the percentage reduction in the number  
15 of test sequences returned by the W-method when it is fed with the W-set  
16 595 of test sequences returned by the proposed algorithm. Let  $T(\cdot)$  be the number of test sequences  
17 returned by the parameter. We will use  $EA(M)$  to denote the test suite returned  
18 given by the parameter. We will use  $EA(M)$  to denote the test suite returned  
19 by the W-method using  $\mathcal{W}_{EA}$  and  $P(M)$  to denote the test suite returned by  
20 the W-method when using  $\mathcal{W}_P$ . Then  $M2$  is computed as follows:  
21  
22  
23

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

24  
25  
26  
27 600 The final measure ( $M3$ ) is the percentage reduction in the total number of  
28 inputs in the test suites returned by the W-Method when using  $\mathcal{W}_P$  and  $\mathcal{W}_{EA}$   
29 respectively.  $M3$  is computed as follows:  
30  
31

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

32  
33 where  $Lt(\cdot)$  returns the sum of the lengths (number of inputs) of the input  
34 sequences in the test suite given in the parameter.  
35  
36  
37  
38

### 39 605 5.5. Results and Evaluation

40  
41 In order to evaluate the relative performance of different approaches, for  
42 each FSM  $M$ , we separately computed W-sets using the proposed algorithm  
43 and the existing algorithm. We then generated test suites using the W-method  
44 and computed the values of the three measures. In these experiments, we used  
45  
46  
47  
48 610 two existing tools to check all of the W-sets and test suites produced during the  
49 experiments. Given an FSM  $M$ , one tool [37] checks that a given set of input  
50 sequences is a W-set for  $M$  and the other tool [45] checks whether a given set  
51 of input sequences defines a checking experiment for  $M$ .  
52  
53

54 We start with the results for the first set of experimental subjects, the ran-  
55 domly generated FSMs in C1. The results are given in Figure 2.  
56 615  
57  
58

1  
2  
3  
4  
5  
6  
7  
8  
9 We observe from Figure 2b that, for the proposed algorithm, the number of  
10 input sequences in the  $W$ -sets increases only very slowly as the number of states  
11 increases. In contrast, the number of input sequences increases quite noticeably  
12 when we use the traditional approach. Recall that the proposed algorithm  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

620 selects the input sequence that distinguishes more pairs than the others while  
constructing the  $W$ -set (Lines 21-25 of Algorithm 1). Therefore, as the number  
of states increases the algorithm has more options for reducing the number of  
input sequences and hence can pick input sequences that distinguish more pairs  
than others. This helps explain why the  $W$ -set size grows more slowly when  
625 using the proposed algorithm.

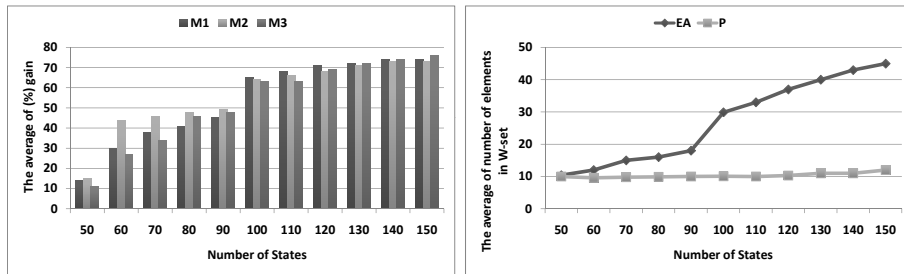
The results, regarding the size and length of the test suites returned by the  
 $W$ -method, are given in Figure 2c and Figure 2d respectively. In both figures,  
we observe that using the  $P$  algorithm leads to test suites with fewer elements  
and fewer inputs than the  $EA$  algorithm.

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

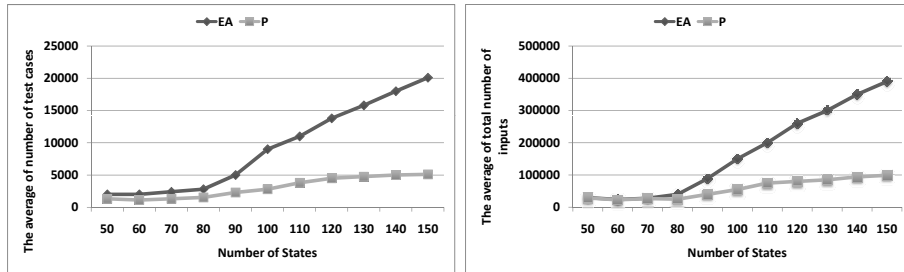
The results for C2 are given in Figure 3. We observe from Figure 3b that  
the sizes of the  $W$ -sets plateaued for the FSMs if the proposed algorithm is  
635 used. In contrast, the mean  $W$ -set size increases steadily with the number of  
states when computed by  $EA$ . This again stems from the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

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

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

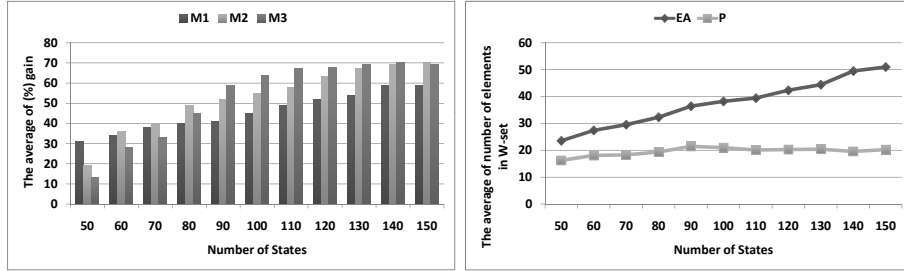


(a) Averages of the gain with respect to metrics  $M1$ ,  $M2$  and  $M3$  from  $C1$ . (b) Averages of the total number of input sequences in  $W$ -sets constructed from  $C1$ .

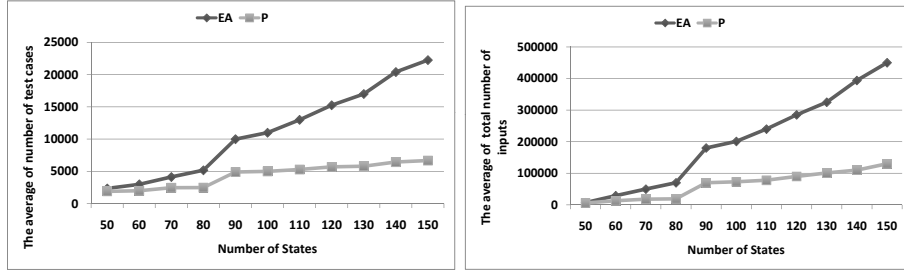


(c) Averages of the total number of input sequences in test suites constructed from  $C1$ . (d) Averages of the total number of inputs in test suites constructed from  $C1$ .

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

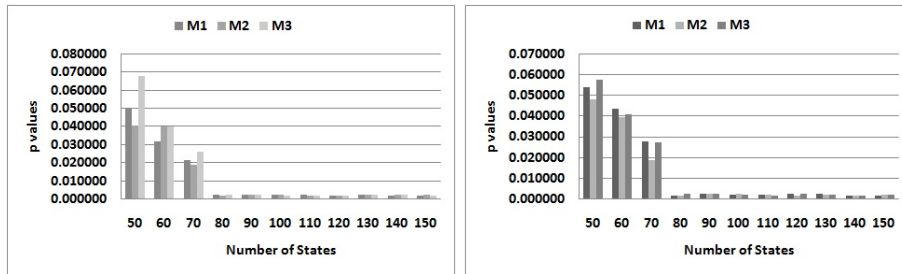


(a) Averages of the gains with respect to metrics  $M1$ ,  $M2$  and  $M3$  from  $C2$ . (b) Averages of the total number of input sequences in  $W$ -sets constructed from  $C2$ .



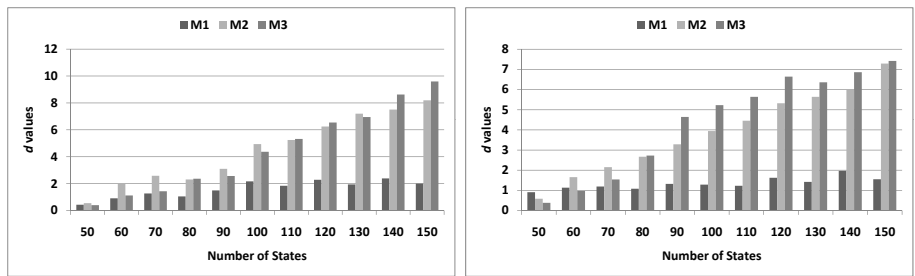
(c) Averages of the total number of input sequences in test suites constructed from  $C2$ . (d) Averages of the total number of inputs in test suites constructed from  $C2$ .

Figure 3: Performance comparison of algorithms  $W_{EA}$  and  $W_P$  on different metrics observed from  $C2$ .



(a) p-values for  $C1$  (b) p-values for  $C2$

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



(a) Cohen's  $d$  metric results on results of metrics  $M1$ ,  $M2$  and  $M3$  retrieved from C1. (b) Cohen's  $d$  metric results on results of metrics  $M1$ ,  $M2$  and  $M3$  retrieved from C2.

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

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

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

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

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

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

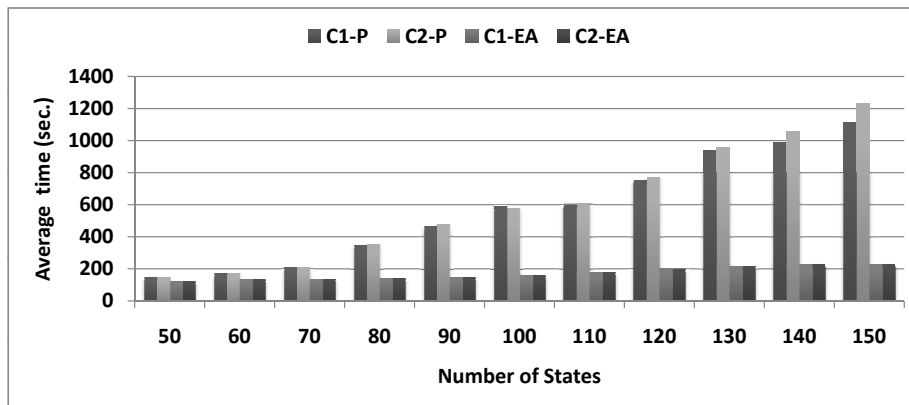


Figure 6: Time comparison of the W-set generation algorithms. X axis labels the number of states, and Y axis labels the time (seconds).

algorithm tends to lead to smaller test suites.

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

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

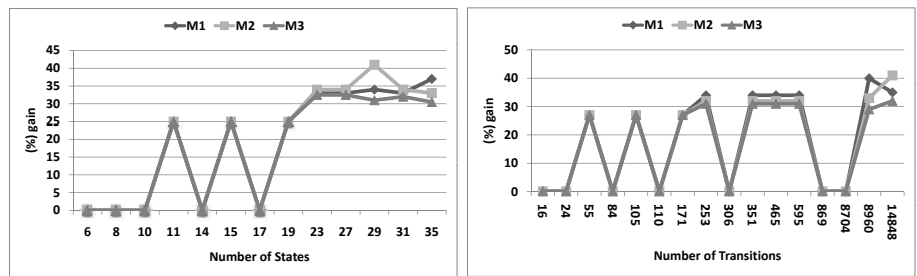
The results of the experiments conducted on the benchmarks are given in Table 3. We see that, for some of the specifications, there is no difference in the number of elements in the  $W$ -sets and the number of test sequences. However, this is not too surprising since these FSMs are relatively small.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

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

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

### 5.6. Threats to validity

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



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

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

Name	M1	M2	M3	P(msec.)	EA(msec.)
Shift Register	0	0	0	25	24
Coffee Machine	0	0	0	< 1	< 1
ABPSender.flat01	25%	25%	25%	1	1
Maestro	0	0	0	< 1	< 1
ABPSender.flat02	25%	25%	25%	1	1
ABPChannelflame.flat.02	0	0	0	1	1
ABPSender.flat03	25%	25%	25%	1	1
ABPSender.flat04	33%	34%	32%	3	2
ABPChannelflame.flat03	0	0	0	1	1
ABPSender.flat05	33%	34%	32%	4	3
ABPSender.flat06	33%	34%	32%	6	7
ABPSender.flat07	33%	34%	32%	15	10
Ex4	0	0	0	270	195
Log	0	0	0	260	234
DVRAM	40%	32%	29%	567	267
Rie	34%	41%	31%	689	304

Table 3: Results of experiments conducted on benchmark FSMs

1  
2  
3  
4  
5  
6  
7  
8  
9 First, there are threats to generalisability. We evaluated the performance of  
10 our algorithm by using randomly generated FSMs. It is possible that the per-  
11 formance of our proposed algorithm differs for FSMs used in real-life situations.  
12  
13  
14  
710 Although using random FSMs is normal in this field, in order to evaluate the gen-  
15 eralization of the proposed algorithm, we also evaluated the proposed algorithm  
16 using case studies consisting of benchmark FSM specifications (Section 5.3.3).  
17 We see that the experimental results obtained from randomly generated FSMs  
18 are similar to the results obtained with the larger benchmark FSMs. How-  
19  
20  
21  
715 ever, for small FSMs, we did not observe any difference between the proposed  
22 algorithm and the existing algorithm.  
23

24  
25 There are also threats to internal validity and the possibility that one or  
26 more of the implementations were incorrect. In order to reduce this threat,  
27 we also used two existing tools [37, 45]. The first tool checks if a given set of  
28 sequences is a W-set for an FSM. The second tool checks whether or not a given  
29  
30  
720 set of input sequences defines a checking experiment for an FSM  $M$ .  
31  
32

## 33 34 **6. Related Work**

35  
36  
37 Minimizing the W-set is inherently related to i) minimizing the size of sep-  
38 arating sequences and ii) test suite minimization.

39  
40  
725 To begin with, to our knowledge, this work is the first to investigate the  
41 problem of finding a small W-set. In [49] the author discusses an algorithm that  
42 constructs W-sets with short sequences. Note that the algorithm presented in  
43 this previous work is identical to the algorithm given in [26].  
44  
45

46  
47  
730 However, for a given FSM there are a number of different types of sequences  
48 that can separate states, with examples including *adaptive distinguishing se-*  
49 *quences* (ADSs), and *unique input output sequences (UIOs)*. Previous work  
50 explored the problem of construct minimal ADSs. It was proven that minimiz-  
51 ing the height of an ADS (in fact minimizing ADS size with respect to some  
52 other metrics as well) is an NP-hard problem [50]. Türker and Yenigün proposed  
53  
54  
55  
735 two heuristics as a modification of the existing ADS generation algorithm for  
56  
57  
58

1  
2  
3  
4  
5  
6  
7  
8  
9 minimization [50]. Recently Türker et al. also presented a BFS based algorithm  
10 called the *lookahead based algorithm* (LA) for minimizing ADSs [51]. In LA, a  
11 branch of a BFS tree is extended if it satisfies certain conditions. However, the  
12 algorithm proposed in this paper expands each branch of a BFS tree until the  
13  
14  
15  
16 740 W-set is constructed. Regarding UIOs, the literature contains several works on  
17 reducing the size of these sequences. First, the algorithm introduced in [9] is a  
18 brute-force algorithm based on a BFS tree. Therefore, this algorithm finds the  
19 shortest such sequences. Later in [52], the author introduced a heuristic to find  
20  
21  
22 UIOs faster.

23  
24 745 The proposed W-set generation algorithm includes a step in which the choice  
25 of input sequence is based on how many pairs of states are separated by the  
26 sequences. The approach taken is a form of greedy algorithm, in which the test  
27 sequence with greatest ‘score’ (number of pairs of states separated) is chosen.  
28 Greedy algorithms have been used in a number of areas of software testing,  
29  
30  
31 750 including two topics relevant to regression testing: test minimization and (later)  
32 test prioritization (see, for example, [53, 54, 55]). The context of this previous  
33 work is that we have a regression test suite  $T$  and wish to either reduce the cost  
34 of applying regression testing by only using a subset of  $T$  (test minimization)  
35 or to apply the test cases in an order that means that any failures are likely to  
36  
37  
38  
39 755 be observed relatively early.

40  
41 One possibility would have been to simply sort the sequences based on how  
42 many pairs of states are distinguished and this would then have been similar to  
43 the Greedy Algorithm that was devised for prioritizing test cases based on cov-  
44 erage information (see, for example, [55]). This prioritization technique involves  
45  
46  
47 760 computing the coverage provided by each test case and then simply sorting the  
48 test cases based on the coverage scores. However, instead, whenever the se-  
49 quence length is increased we use an updated set  $\Delta$  of pairs of states not yet  
50 covered and the choice of next input sequence is based on the updated  $\Delta$ . This  
51 is a little like the so-called Additional Greedy approach to test prioritisation.  
52  
53  
54  
55 765 Here, in each iteration, the approach chooses the test case providing highest  
56 additional coverage and then updates the coverage information [55].  
57  
58  
59  
60  
61  
62  
63  
64  
65

1  
2  
3  
4  
5  
6  
7  
8  
9       Although Additional Greedy has been found to be an effective prioritiza-  
10       tion technique, it has been observed that it can be sub-optimal. As a result,  
11       a number of alternatives have been explored. In particular, it has been found  
12       that metaheuristic algorithms, such as Genetic Algorithms, can outperform Ad-  
13       770       ditional Greedy (see, for example, [56, 57]). It may well be that the proposed  
14       algorithm can be improved by incorporating such algorithms. However, such  
15       approaches are likely to significantly increase the (algorithm) execution time  
16       and so probably will only be of value in situations in which the benefits of a  
17       small reduction in test suite size are not outweighed by such an increase in W-set  
18       775       generation time.  
19  
20  
21  
22  
23  
24  
25

## 26       7. Conclusion

27  
28       Software testing is typically performed manually and is an expensive, error  
29       prone process. This has led to interest in automated test generation, includ-  
30       780       ing significant interest in model based testing (MBT). Most MBT techniques  
31       generate test suites from either finite state machines (FSMs) or labelled tran-  
32       sition systems, with such a model potentially representing the semantics of a  
33       specification in a richer language.  
34  
35  
36  
37

38       In this paper, we investigated the problem of computing a minimal W-set  
39       785       for a given deterministic, minimal completely specified FSM. We introduced the  
40       associated decision problem and showed that the problem of deciding whether  
41       an FSM has a W-set with at most  $K$  input sequences is PSPACE-complete. In  
42       contrast, the problem of deciding whether an FSM has a W-set with total length  
43        $K$  is NP-complete.  
44  
45  
46

47       790       The initial motivation for minimizing a W-set was the use of W-sets in the  
48       context of test suite generation. Ideally one uses a minimal W-set, since the  
49       W-set is used in state recognition and state verification. Therefore, the size of a  
50       test suite generated using a W-set should correlate with the number of elements  
51       of the W-set. Due to the hardness of W-set minimization, a heuristic algorithm  
52       795       was proposed. Experiments were conducted to evaluate the proposed algorithm.  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

1  
2  
3  
4  
5  
6  
7  
8  
9 In these, the use of a  $W$ -set returned by the proposed algorithm reduced the  
10 number of test sequences in a test suite by 37.3% on average, with the total  
11 number of inputs being reduced by 36.4% on average.  
12

13  
14 Although the proposed algorithm was evaluated in the context of the  $W$ -  
15 method, there are potential implications for some other FSM-based test gener-  
16 800 ation techniques. The  $W_p$  [9], the HSI-method [28], the HIS [58]<sup>7</sup>, H [35, 32],  
17 and incremental methods (such as [59]) rely on the presence of a  $W$ -set and  
18 one might conjecture that the use of a small  $W$ -set is likely to lead to these  
19 returning more efficient test suites.  
20  
21  
22

23 805 There are a number of additional lines of future work. First, it would be  
24 interesting to explore realistic conditions under which the decision and optimisa-  
25 tion problems can be solved in polynomial time. Such conditions might lead to  
26 new notions of testability. Although the results of the experiments suggest that  
27 the use of relatively small  $W$ -sets leads to test suites that require fewer resets,  
28 it would be interesting to extend the experiments and possibly also consider  
29 810 the  $W_p$ , HIS and SPY algorithms [9, 58, 22]. Finally, it would be interest-  
30 ing to investigate complexity results and effective algorithms that can generate  
31 minimum  $W$ -sets from non-deterministic FSMs.  
32  
33  
34  
35  
36  
37  
38

### 39 Acknowledgement

40  
41 815 This work is dedicated to the memory of Prof. Hasan Ural, who was our  
42 mentor/friend. Prof. Ural was a good teacher who introduced new scientists to  
43 the testing community, he was also a dedicated scientist who provided important  
44 contributions relentlessly to the testing spectra for nearly half a century.  
45  
46  
47

---

48  
49 <sup>7</sup>Note, we use the naming convention used in [32] and use the HIS method to refer to the  
50 test suite generation algorithm.  
51  
52  
53  
54  
55  
56  
57  
58

1  
2  
3  
4  
5  
6  
7  
8  
9 **References**

- 10  
11 820 [1] A. Friedman, P. Menon, Fault detection in digital circuits, Computer Ap-  
12 plications in Electrical Engineering Series, Prentice-Hall, 1971.  
13  
14  
15 [2] A. Aho, R. Sethi, J. Ullman, Compilers, principles, techniques, and tools,  
16 Addison-Wesley series in computer science, Addison-Wesley Pub. Co., 1986.  
17  
18  
19 [3] T. S. Chow, Testing software design modelled by finite state machines,  
20 825 IEEE Transactions on Software Engineering 4 (1978) 178–187.  
21  
22  
23 [4] E. Brinksma, A theory for the derivation of tests, in: Proceedings of Pro-  
24 tocol Specification, Testing, and Verification VIII, North-Holland, Atlantic  
25 City, 1988, pp. 63–74.  
26  
27  
28 [5] A. Dahbura, K. Sabnani, M. Uyar, Formal methods for generating protocol  
29 830 conformance test sequences, Proceedings of the IEEE 78 (8) (Aug) 1317–  
30 1326.  
31  
32  
33 [6] D. Lee, K. Sabnani, D. Kristol, S. Paul, Conformance testing of protocols  
34 specified as communicating finite state machines-a guided random walk  
35 based approach, IEEE Transactions on Communications 44 (5) (May) 631–  
36 640.  
37 835  
38  
39 [7] D. Lee, M. Yannakakis, Principles and methods of testing finite-state ma-  
40 chines - a survey, Proceedings of the IEEE 84 (8) (1996) 1089–1123.  
41  
42  
43 [8] S. Low, Probabilistic conformance testing of protocols with unobservable  
44 transitions, in: 1993 International Conference on Network Protocols, Oct,  
45 840 pp. 368–375.  
46  
47  
48 [9] K. Sabnani, A. Dahbura, A protocol test generation procedure, Computer  
49 Networks 15 (4) (1988) 285–297.  
50  
51  
52 [10] D. P. Sidhu, T.-K. Leung, Formal methods for protocol testing: A detailed  
53 study, IEEE Transactions on Software Engineering 15 (4) (1989) 413–426.  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

- 1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65
- 845 [11] R. V. Binder, *Testing Object-Oriented Systems: Models, Patterns, and Tools*, Addison-Wesley, 1999.
- [12] M. Haydar, A. Petrenko, H. Sahraoui, Formal verification of web applications modeled by communicating automata, in: *Formal Techniques for Networked and Distributed Systems FORTE*, Vol. 3235 of LNCS, Springer-Verlag, Madrid, 2004, pp. 115–132.
- 850 [13] A. Betin-Can, T. Bultan, Verifiable concurrent programming using concurrency controllers, in: *Proceedings of the 19th IEEE international conference on Automated software engineering*, IEEE Computer Society, 2004, pp. 248–257.
- [14] I. Pomeranz, S. M. Reddy, Test generation for multiple state-table faults in finite-state machines, *IEEE Transactions on Computers* 46 (7) (1997) 783–794.
- 855 [15] M. Utting, A. Pretschner, B. Legeard, A taxonomy of model-based testing approaches, *Software Testing, Verification and Reliability* 22 (5) (2012) 297–312.
- 860 [16] W. Grieskamp, N. Kicillof, K. Stobie, V. A. Braberman, Model-based quality assurance of protocol documentation: tools and methodology (2011).
- [17] A. V. Aho, A. T. Dahbura, D. Lee, M. U. Uyar, An optimization technique for protocol conformance test generation based on UIO sequences and rural chinese postman tours, in: *Protocol Specification, Testing, and Verification VIII*, Elsevier (North-Holland), Atlantic City, 1988, pp. 75–86.
- 865 [18] F. C. Hennie, Fault-detecting experiments for sequential circuits, in: *Proceedings of Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, Princeton, New Jersey, 1964, pp. 95–110.
- [19] G. Gonenc, A method for the design of fault detection experiments, *IEEE Transactions on Computers* 19 (1970) 551–558.

- 1  
2  
3  
4  
5  
6  
7  
8  
9 [20] S. T. Vuong, W. W. L. Chan, M. R. Ito, The UIOv-method for protocol  
10 test sequence generation, in: The 2nd International Workshop on Protocol  
11 Test Systems, Berlin, 1989.  
12  
13  
14  
15 875 [21] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, A. Ghedamsi,  
16 Test selection based on finite state models, *IEEE Transactions on Software*  
17 *Engineering* 17 (6) (1991) 591–603.  
18  
19  
20 [22] A. da Silva Simão, A. Petrenko, N. Yevtushenko, On reducing test length  
21 for FSMs with extra states, *Software Testing, Verification and Reliability*  
22 22 (6) (2012) 435–454.  
23 880  
24  
25 [23] H. Ural, K. Zhu, Optimal length test sequence generation using distin-  
26 guishing sequences, *IEEE/ACM Transactions on Networking* 1 (3) (1993)  
27 358–371.  
28  
29  
30 [24] A. Petrenko, N. Yevtushenko, Testing from partial deterministic FSM spec-  
31 ifications, *IEEE Transactions on Computers* 54 (9) (2005) 1154–1165.  
32 885  
33  
34 [25] Z. Kohavi, *Switching and Finite State Automata Theory*, McGraw-Hill,  
35 New York, 1978.  
36  
37  
38 [26] A. Gill, *Introduction to The Theory of Finite State Machines*, McGraw-Hill,  
39 New York, 1962.  
40  
41  
42 890 [27] M. P. Vasilevskii, Failure diagnosis of automata, *Cybernetics* 4 (1973) 653–  
43 665.  
44  
45  
46 [28] G. Luo, A. Petrenko, G. V. Bochmann, Selecting test sequences for  
47 partially-specified nondeterministic finite state machines, in: *Protocol Test*  
48 *Systems*, IFIP The International Federation for Information Processing,  
49 Springer US, 1995, pp. 95–110.  
50 895  
51  
52 [29] D. Angulin, Learning regular sets from queries and counterexamples, *In-*  
53 *formation and Computation* 75 (1987) 87–106.  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

- 1  
2  
3  
4  
5  
6  
7  
8  
9 [30] D. Huistra, J. Meijer, J. van de Pol, Adaptive learning for learn-based re-  
10 regression testing, in: Formal Methods for Industrial Critical Systems, Vol.  
11 11119 of Lecture Notes in Computer Science, Springer International Pub-  
12 900 lishing, Cham, 2018, pp. 162–177.  
13  
14  
15  
16 [31] N. Yang, K. Aslam, R. Schiffelers, L. Lensink, D. Hendriks, L. Cleophas,  
17 A. Serebrenik, Improving model inference in industry by combining active  
18 and passive learning, in: 2019 IEEE 26th International Conference on Soft-  
19 ware Analysis, Evolution and Reengineering (SANER), 2019, pp. 253–263.  
20 905  
21  
22  
23 [32] R. Dorofeeva, K. El-Fakih, N. Yevtushenko, An improved conformance  
24 testing method, in: Proceedings of the 25th IFIP WG 6.1 International  
25 Conference on Formal Techniques for Networked and Distributed Systems,  
26 FORTE’05, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 204–218.  
27  
28  
29  
30 910 [33] A. Petrenko, G. v. Bochmann, R. Dssouli, Conformance relations and test  
31 derivation, in: Proceedings of Protocol Test Systems VI (C-19), 1993, pp.  
32 157–178.  
33  
34  
35 [34] A. D. Friedman, P. R. Menon, Fault detection in digital circuits, Prentice-  
36 Hall Englewood Cliffs, N.J, 1971.  
37  
38  
39 915 [35] I. Koufareva, M. Dorofeeva, A novel modification of W-method, Joint Bull.  
40 Novosibirsk Comput (2002) 69–81.  
41  
42  
43 [36] E. P. Hsieh, Checking experiments for sequential machines, IEEE Transac-  
44 tions on Computers 20 (1971) 1152–1166.  
45  
46  
47 [37] R. M. Hierons, U. C. Türker, Parallel algorithms for generating harmonised  
48 state identifiers and characterising sets, IEEE Transactions on Computers  
49 920 65 (11) (2016) 3370–3383.  
50  
51  
52 [38] R. M. Hierons, Minimizing the number of resets when testing from a finite  
53 state machine, Information Processing Letters 90 (6) (2004) 287–292.  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

- 1  
2  
3  
4  
5  
6  
7  
8  
9 [39] R. Dorofeeva, K. El-Fakih, S. Maag, A. R. Cavalli, N. Yevtushenko, FSM-  
10 based conformance testing methods: a survey annotated with experimental  
11 925 evaluation, *Information and Software Technology* 52 (12) (2010) 1286–1297.  
12  
13  
14 [40] K. Bulut, G. Jourdan, U. C. Türker, Minimizing characterizing sets, in:  
15 16th International Conference on Formal Aspects of Component Software  
16 (FACS 2019), Vol. 12018 of Lecture Notes in Computer Science, Springer,  
17 2019, pp. 72–86.  
18 930  
19 [41] D. Lee, M. Yannakakis, Testing finite-state machines: State identification  
20 and verification, *IEEE Transactions on Computers* 43 (3) (1994) 306–320.  
21  
22 [42] M. R. Garey, D. S. Johnson, *Computers and Intractability*, W. H. Freeman  
23 and Company, New York, 1979.  
24  
25 [43] F. Brglez, ACM/SIGMOD benchmark dataset, Available on-  
26 935 line at [https://people.engr.ncsu.edu/brglez/CBL/benchmarks/  
27 Benchmarks-upto-1996.html](https://people.engr.ncsu.edu/brglez/CBL/benchmarks/Benchmarks-upto-1996.html), accessed: 2014-02-13 (1996).  
28  
29 [44] D. Neider, R. Smetsers, F. W. Vaandrager, H. Kuppens, Benchmarks for  
30 automata learning and conformance testing, Vol. 11200 of Lecture Notes  
31 in Computer Science, Springer, 2018, pp. 390–416.  
32 940  
33 [45] C. Güniçen, U. C. Türker, H. Ural, H. Yenigün, Generating preset distin-  
34 guishing sequences using SAT, in: *Computer and Information Sciences II*,  
35 Springer London, 2012, pp. 487–493.  
36  
37 [46] F. Wilcoxon, *Individual Comparisons by Ranking Methods*, Springer New  
38 York, New York, NY, 1992, pp. 196–202.  
39 945  
40 [47] P. Teetor, *R Cookbook*, 1st Edition, O’Reilly, 2011.  
41  
42 [48] J. Cohen, *Statistical power analysis for the behavioral sciences*, Routledge,  
43 1988.  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

- 1  
2  
3  
4  
5  
6  
7  
8  
9 [49] M. Soucha, Finite-state machine state identification sequences, Avail-  
10 able online at <https://cyber.felk.cvut.cz/theses/papers/555.pdf>,  
11 950 accessed: 2020-12-06 (1996).  
12  
13  
14 [50] U. Türker, H. Yenigün, Hardness and inapproximability of minimizing  
15 adaptive distinguishing sequences, *Formal Methods in System Design* 44 (3)  
16 (2014) 264–294.  
17  
18  
19 [51] U. Türker, T. Ünlüyurt, H. Yenigün, Lookahead-based approaches for mini-  
20 955 [51] U. Türker, T. Ünlüyurt, H. Yenigün, Lookahead-based approaches for mini-  
21 mizing adaptive distinguishing sequences, in: *Testing Software and Systems*  
22 - 26th IFIP WG 6.1 International Conference, ICTSS 2014, Madrid, Spain,  
23 September 23-25, 2014. *Proceedings*, 2014, pp. 32–47.  
24  
25  
26 [52] K. Naik, Efficient computation of unique input/output sequences in finite-  
27 state machines, *IEEE/ACM Transactions on Networking* 5 (4) (1997) 585–  
28 960 599.  
29  
30  
31  
32 [53] T. Y. Chen, M. F. Lau, Dividing strategies for the optimization of a test  
33 suite, *Information Processing Letters* 60 (3) (1996) 135 – 141.  
34  
35  
36 [54] G. Rothermel, M. J. Harrold, Analyzing regression test selection tech-  
37 niques, *IEEE Transactions on Software Engineering* 22 (8) (1996) 529–551.  
38 965  
39  
40 [55] G. Rothermel, M. J. Harrold, J. von Ronne, C. Hong, Empirical studies of  
41 test-suite reduction, *Software Testing, Verification and Reliability* 12 (4)  
42 (2002) 219–249.  
43  
44  
45 [56] Z. Li, M. Harman, R. M. Hierons, Search algorithms for regression test case  
46 prioritization, *IEEE Transactions on Software Engineering* 33 (4) (2007)  
47 970 225–237.  
48  
49  
50 [57] S. Yoo, M. Harman, Regression testing minimization, selection and pri-  
51 oritization: a survey, *Software Testing, Verification and Reliability* 22 (2)  
52 (2012) 67–120.  
53  
54  
55  
56  
57  
58

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65

<sup>975</sup> [58] A. Petrenko, N. Yevtushenko, A. Lebedev, A. Das, Nondeterministic state machines in protocol conformance testing, in: Proceedings of Protocol Test Systems, VI (C-19), Elsevier Science (North-Holland), Pau, France, 1994, pp. 363–378.

[59] K. El-Fakih, R. Dorofeeva, N. Yevtushenko, G. V. Bochmann, FSM-based <sup>980</sup> testing from user defined faults adapted to incremental and mutation testing, Programming and Computer Software 4/38 (2012) 1608–3261.