Poster: An Experimental Analysis of Fault Detection Capabilities of Covering Array Constructors

Rubing Huang, Yunan Zhou
Jiangsu University, China
rbhuang@ujs.edu.cn

Tsong Yueh Chen
Swinburne University of Technology, Australia
tychen@swin.edu.au

Dave Towey
University of Nottingham Ningbo China, China
dave.towey@nottingham.edu.cn

Jinfu Chen
Jiangsu University, China
jinfuchen@ujs.edu.cn

ABSTRACT

Combinatorial Interaction Testing (CIT) aims at constructing an effective test suite, such as a *Covering Array* (CA), that can detect faults that are caused by the interaction of parameters. In this paper, we report on some empirical studies conducted to examine the fault detection capabilities of five popular CA constructors: `ACTS`, `Jenny`, `PICT`, `CASA`, and `TCA`. The experimental results indicate that Jenny has the best performance, because it achieves better fault detection than the other four constructors in many cases. Our results also indicate that CAs generated using `ACTS`, `PICT`, or `CASA` should be prioritized before testing.

CCS CONCEPTS

• **Software and its engineering** → *Software testing and debugging*;

KEYWORDS

Combinatorial Interaction Testing, Covering Array, Constructor, Empirical Study, Software Testing

ACM Reference Format:
Rubing Huang, Yunan Zhou, Tsong Yueh Chen, Dave Towey, and Jinfu Chen. 2018. Poster: An Experimental Analysis of Fault Detection Capabilities of Covering Array Constructors. In *ICSE '18 Companion: 40th International Conference on Software Engineering Companion, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages.

1 INTRODUCTION

Combinatorial Interaction Testing (CIT) [6] provides a trade-off between testing efficacy and efficiency, and aims at constructing an effective test suite (such as a *Covering Array* (CA) [6]) to detect faults caused by parameter interactions.

Many algorithms exist to support the construction of covering arrays according to different strategies [6]. However, to the best of our knowledge, there has not yet been any study to compare the fault detection capabilities among different CA constructors. Therefore, in this paper we report a comparison of five popular, publicly available CA constructors that can support constraints on parameter values. Experiments were carried out to examine the fault detection capability of each CA construction tool, the results of which are presented here.

2 EXPERIMENTS

In our study, we examined five versions of each of two C programs: *flex* and *grep*. The *flex* program is a lexical analysis generator, and *grep* is a widely-used command-line tool for searching and processing text, matching regular expressions.

Although many tools exist to support the construction of CAs, our study utilized the following five representative tools with different implementation mechanisms: `ACTS` [4], `Jenny` [1], `PICT` [2], `CASA` [3], and `TCA` [5]. We focused on the generation strengths τ = 2, 3, 4, and 5. Different CA constructors can generate τ-wise CAs with different sizes for each program. In our experiments, we identified the smallest CA size (k) for each program at each strength τ — which involved examining 200 CAs for each constructor, for each of four programs, giving a total of 4000 CAs per τ value. To ensure a fair comparison, we then only executed a total of k test cases for each CA.

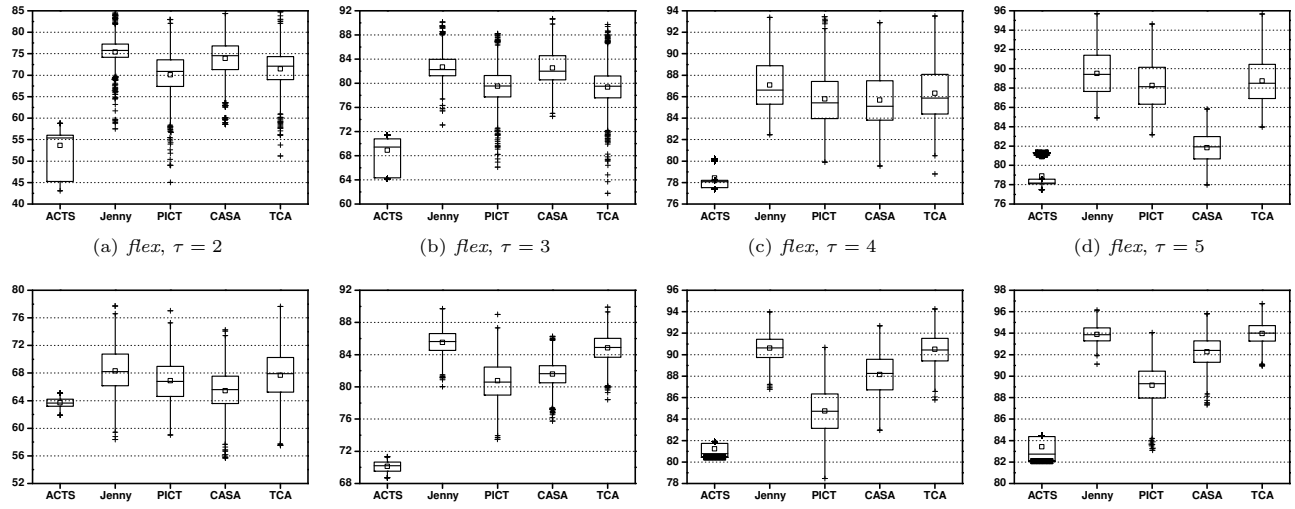
Table 1 presents the mean *fault detection effectiveness* (FDE) results (measured as the percentage of faults detected), with the largest values highlighted in bold and italics. Figure 1 shows the *average percentage of faults detected* (APFD) [7], with each subfigure showing the distribution of APFD values for a particular program and strength, broken down according to CA constructor — each of which has 200

Table 1: FDE Results

Object	Constructor	$\tau = 2$	$\tau = 3$	$\tau = 4$	$\tau = 5$
flex	`ACTS`	87.21	82.55	***96.03***	***95.91***
	`Jenny`	***87.32***	***90.11***	92.29	94.50
	`PICT`	87.21	89.79	92.39	94.07
	`CASA`	86.98	89.86	92.56	94.19
	`TCA`	87.31	89.76	92.35	94.07
grep	`ACTS`	***88.23***	93.92	95.91	***99.97***
	`Jenny`	86.29	95.07	98.36	***99.97***
	`PICT`	86.21	***95.11***	97.56	***99.97***
	`CASA`	85.99	94.75	98.39	99.96
	`TCA`	85.90	94.87	***98.65***	***99.97***

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3194953



(a) *flex*, $\tau = 2$ (b) *flex*, $\tau = 3$ (c) *flex*, $\tau = 4$ (d) *flex*, $\tau = 5$

(e) *grep*, $\tau = 2$ (f) *grep*, $\tau = 3$ (g) *grep*, $\tau = 4$ (h) *grep*, $\tau = 5$

Figure 1: APFD results

data points. Based on the experimental results, we have the following observations:

(1) In terms of FDE results, each CA constructor may have different performances for different programs — in other words, each constructor may have the highest FDE results for some CAs, but may also have the lowest results for some other CAs. Overall, however, `ACTS` and `Jenny` perform slightly better than other constructors, even though they may also have the worst performances in a few cases. Nevertheless, the differences in fault detection capability are small, especially when the generation strength is high.

(2) In terms of APFD results, `ACTS` generally performs worst, and `Jenny` has the best fault detection rates, followed by `TCA`. Additionally, either `PICT` or `CASA` could achieve the second worst performance for some cases.

3 CONCLUSIONS

In this work, we empirically examined five representative CA constructors, in terms of fault detection capability. To conclude, we have the following findings:

(1) There is no CA constructor that is always best for all programs.

(2) From the perspective of fault detection, `ACTS` and `Jenny` can detect slightly more faults than other CA constructors, but the performance difference among all constructors is small in most cases. Nevertheless, `Jenny` has the best rates of fault detection in many cases, followed by `TCA`; while `ACTS` has the worst, followed by `PICT` and `CASA`.

(3) Overall, when CIT testers have no initial preference, then `Jenny` is recommended. However, if a decision has been made to use `ACTS`, `PICT`, or `CASA`, then their constructed CAs should be prioritized before testing, which should improve their APFD performance.

Due to the space limitations, we could only report the results for two subject programs. A larger scale empirical study is under way to compare CA constructors using more programs and more evaluation metrics.

ACKNOWLEDGMENTS

This work is partially supported by the National Natural Science Foundation of China under Grant Nos. 61502205 and 61202110, the Postdoctoral Science Foundation of China under Grant Nos. 2015M581739 and 2015M571687, the Natural Science Foundation of the Jiangsu Higher Education Institutions of China under Grant No. 15KJB520007, and the Senior Personnel Scientific Research Foundation of Jiangsu University under Grant No. 14JDG039. This work is also supported by the Young Backbone Teacher Cultivation Project of Jiangsu University, and the sponsorship of Jiangsu Overseas Visiting Scholar Program for University Prominent Young & Middle-aged Teachers and Presidents.

REFERENCES

[1] `Jenny`: A freely available CA constructor. `http://burtleburtle.net/bob/math/jenny.html`.

[2] J. Czerwonka. 2006. Pairwise Testing in Real World: Practical Extensions to Test Case Generators. In *Proceedings of the 24th Pacific Northwest Software Quality Conference (PNSQC'06)*. 419–430.

[3] B. J. Garvin, M. B. Cohen, and M. B. Dwyer. 2011. Evaluating improvements to a meta-heuristic search for constrained interaction testing. *Empirical Software Engineering* 16, 1 (2011), 61–102.

[4] Y. Lei, R. Kacker, D. R. Kuhn, and V. Okun. 2008. IPOG/IPOD: Efficient Test Generation for Multi-Way Software Testing. *Software Testing, Verification, and Reliability* 18, 3 (2008), 125–148.

[5] J. Lin, C. Luo, S. Cai, K. Su, D. Hao, and L. Zhang. 2015. TCA: An Efficient Two-Mode Meta-Heuristic Algorithm for Combinatorial Test Generation. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE'15)*. 494–505.

[6] C. Nie and H. Leung. 2011. A survey of combinatorial testing. *ACM Computer Survey* 43, 2 (2011), 11:1–11:29.

[7] G. Rothermel, R. H. Untch, Chengyun Chu, and M. J. Harrold. 2001. Prioritizing Test Cases for Regression Testing. *IEEE Transactions on Software Engineering* 27, 10 (2001), 929–948.