

GREEDY INTERACTION ELEMENTS COVERAGE ANALYSIS FOR AI-BASED T-WAY STRATEGIES

AbdulRahman A. Al-Sewari¹, Norazlina Khamis², Kamal Z. Zamli^{3*}

¹ Faculty of Engineering and Information Technology,
Dar-Alsalam International University for Science and Technology
Sana'a, Yemen,

²Department of Software Engineering,
Faculty of Computer Science and Information Technology,
Universiti Malaya,

50603 Kuala Lumpur, Malaysia

³Faculty of Computer Science and Software Engineering,
Universiti Malaysia Pahang,

Lebuhraya Tun Razak,
26300 Kuantan, Pahang, Malaysia

*Corresponding Author. Email: kamalz@ump.edu.my Tel: +(609)-5942544

ABSTRACT

Recently, many researchers have started to adopt Artificial Intelligence AI-based strategies for t-way testing. Here, each interaction is covered at most once whenever possible. In many AI-based strategies, sampling for the most optimal test cases is given utmost priority, but measuring of the interaction coverage metric per test case is often neglected. In the situation where not all test cases can be executed due to constraints on project deadline, the availability of interaction coverage metric per test case can be a useful indicator on how greedy each AI-based strategy of interests is. In this manner, test engineers can make informed decision on the selection of suitable strategies for use. In this paper, this study presents a systematic analysis of existing AI-based strategies including that of Hill Climbing HC, Simulated Annealing SA, Tabu Search TS, Great Flood GF, Particle Swarm Optimization PSTG and Harmonic Search Strategy HSS as far as its rate of coverage per test case is concerned. In doing so, this paper demonstrates that HSS, in most cases, gives competitive interaction coverage rate as compared to competing AI-based strategies but with less number of iterations.

Keywords: *Software Engineering, Interaction Testing, T-Way Testing, Harmony Search Algorithm.*

1.0 INTRODUCTION

Modern software is designed to support various platform and environment. In many cases, the same software is being installed in different type of devices running on different platforms and with different hardware setup. As such, software is developed so that it can be customized to suit different user requirements and needs. In order to accommodate such requirements, software engineers have to develop more flexible and highly configurable software. Nonetheless, as software become highly configurable, they tend to be more complicated and complex in term of design, number of inputs, size and implementation. Here, there are potentially many intertwined dependencies between various parameter inputs, thus, exposing the more possibilities of software faults [1].

Software testing is perhaps one of the most important activities within the software development lifecycle. Finding defects as well as ensuring conformance against specification are amongst the main reasons for software testing. Despite its importance in terms of establishing quality, most of the techniques use in software testing has not changed much. In fact, some techniques even dated back in early 80s. Back then, most significant program were less than 10,000 lines of codes. In those days, even in absence of proper documentations, it may still be possible for trained testers to develop test suite by reading the whole program line-by-line, identifying all variables and their interactions in order to trace the key paths of the program. Today, it is common to find program with a few million lines of code, often resulted from the reuse of commercial-of-the-shelf components. Due to its complexity, no individual can master the internals of the program implementation completely in order to permit effective analysis and testing. In fact, the size of test suite for all combinations of inputs also grows significantly with lines of code making exhaustive testing prohibitively and practically impossible [1]. For these reasons, much research is now focusing on a sampling technique based on interaction testing (termed t-way strategy).

Complementing existing testing strategies such as equivalence partitioning and boundary value analysis, interaction testing focuses on assessing the faults caused the t-way interaction of variables, where t indicates the interaction strength. The rationale for t-way testing stemmed from the fact that from empirical observation, the number of input variables involved in software failures is relatively small like in the order of 3 to 6, in some classes of software. If t or fewer variables are known to cause fault, test cases can be generated or sampled on some t-way combinations resulting into a smaller set of test cases for consideration.

Considering the aforementioned test case generation as optimization problem, many researchers have started to adopt Artificial intelligence AI-based strategies for t-way testing. Often, AI-based strategies give optimal results with minimum number of test cases as compared to conventional counterparts [2-7]. Although useful, some issues remain:

- In search of the most optimal AI-based strategies, many researchers have not put sufficient focus on measuring of the interaction coverage metric per test case [2]. Here, measuring the interaction coverage metric per test case is useful to schedule and prioritize test case execution and hence, facilitate fault detection as early as possible. In the case when test engineers are given tight deadlines when too many tests need to be executed, test cases with higher coverage should be executed first as they covered the most interactions. Faults and defects can be expected to be exposed earlier if test cases with higher interaction coverage are executed first.
- When the number of parameter and values are large, it is difficult to verify the correctness of the generated test cases in the given test suite as far as covering all the required t-way interactions is concerned. It could be that failure to cover a particular interaction may result into undesirable consequences.
- Apart from the optimal test size and execution time, test engineers may find it difficult to select myriad of existing AI-based strategies suitable for their cost-effective usage.

Addressing some of the aforementioned issues, this work presents a systematic analysis of existing AI-based strategies including that of Hill Climbing HC [9], Simulated Annealing SA [2], Tabu Search TS [10], Great Flood GF, Particle Swarm Optimization PSTG) [3-5] and Harmonic Search Strategy HSS [6-8]. Here, this paper mainly aim to compare how greedy each AI-based strategy is based on the rate of coverage per test case using known benchmark configurations as published in [11] and [12]. In doing so, this study demonstrates that HSS, in most cases, gives competitive interaction coverage rate as compared to competing AI-based strategies but with less number of iterations.

The rest of this paper is organized as follows. Section 2 highlights some of the related work including from that of pure computational to AI-based strategies. Section 3 gives an overview on the systematic interaction element coverage analysis. Section 4 highlights our experimental results and discussion. Finally, section 5 elaborates our conclusion.

2.0 RELATED WORK

In search for the most optimal strategies, researchers have resorted to combinations of many different approaches. As the t-way test generation problem is NP hard [13, 14], it appears impossible for a single strategy to always generate optimal results all the time.

Existing t-way strategies can be categorized as either pure computational or AI- based strategies. In the first case, the strategy use pure computational algorithms to search and sample for optimal test suite. As the name suggest, the latter case adopts the well-known AI-based optimization algorithm. Implementation wise, both computational and AI-based strategies take either one-parameter-at-a-time or one-test- at-a-time approach to generate the final test suite.

One-parameter-at-a-time approach is pioneered by In-Parameter-Order-Generator IPOG [15, 16] and its families of strategies including that of IPOD [13], IPOF [16], IPOF2 [16], and MIPOG [17]. IPOG starts with a test set for the first two parameters, and then extends the test set by generating the pair for the first three parameters and so on, until all the system parameters are covered. This is followed by vertical extension to cover the uncovered interactions, if necessary.

In the case of one-test-at-a-time approach, the corresponding strategy often perform iterative searches through all interaction elements and always constructs a complete test case per iteration until completion such as all interaction elements are covered. Examples of strategies adopt this approach include Automatic Efficient Test Generator AETG [18], TConfig [19], Test Vector Generator TVG [20, 21], Jenny [22], and GTWay [1].

As elaborated earlier, AI-based strategies also adopt one-test-at-a-time approach. The difference between AI-based strategies and conventional one-test-at-a-time strategies lie on the search algorithm employed. As the name suggest, AI-based strategies adopts AI-based algorithms to perform the searching process. Currently, AI-based algorithms such as Hill Climbing HC [9], Simulated Annealing SA [2], Tabu Search TS [10], Great Flood GF, Particle Swarm Optimization PSO [3-5], and Harmony Search Algorithm [6-8] have been successfully adopted for constructing interaction test suites.

Comparatively, strategies that adopt AI-based algorithms often achieve better results in terms of getting the most optimal test suite size in most cases; whereas, strategies that summon pure computational algorithms achieve better generation time in most cases [9, 23, 24].

3.0 INTERACTION ELEMENTS COVERAGE ANALYSIS

In order to understand how the interaction test suites are constructed and how the interaction elements are covered, this paper adopts a simple pizza ordering system refer to Fig. 1. Table 1 represents the system with five input parameters, Pizza Type, Crust, Delivery, Toppings and Size.

The screenshot shows a 'Pizza Ordering' window with the following options selected:

- Pizza Type:** Vegetarian Cheese
- Crust:** Thin Crust
- Toppings:** Roasted Chicken
- Size:** Large
- Delivery:** Eat In

Available options for each dropdown are:

- Pizza Type:** Vegetarian Cheese, Meat Lover
- Crust:** Thin Crust, Extra Thick
- Toppings:** Roasted Chicken, Ground Beef, Mushroom
- Size:** Large, Medium, Small
- Delivery:** Eat In, Take Away

Buttons: Confirm, Reset, Cancel

Fig. 1. Pizza Ordering System

Here, symbolic parameters and values are used instead of real data to ease the discussion. The symbolic parameters and values assignments are denoted by the equality symbol \approx as depicted in Table 1.

PizzaType (P1) \approx {VC, ML}
 Crust (P2) \approx {TC, ET}
 Toppings (P3) \approx {RC, GB, Mu}
 Size (P4) \approx {La, Me, Sm}
 Delivery (P5) \approx {EI, TA}

Table 1. Pizza Options Parameters

Pizza Type P1	Crust P2	Toppings P3	Size P4	Delivery P5
Vegetarian Cheese \approx VC	Thin Crust \approx TC	Roasted Chicken \approx RC	Large \approx La	Eat In \approx EI
Meat Lover \approx ML	Extra Thick \approx ET	Ground Beef \approx GB	Medium \approx Me	Take Away \approx TA
		Mushroom \approx Mu	Small \approx Sm	

The exhaustive test suite or the full interaction strength with $t = 5$ of the pizza ordering system is achieved using $2 \times 2 \times 3 \times 3 \times 2 = 72$ test cases.

When the number of parameters and values grow, this exhaustive test suite grows exponentially. Assuming that here in this example the testers are looking for pairwise interaction with $t = 2$, the number of test cases can be reduced significantly, and with full coverage of the pairwise interaction elements.

The interaction elements are constructed by taking all the interactions of system parameters then assigning the related values for each interaction. Table 2 depicts all the pairwise interaction elements for the pizza ordering system configuration. Here, all pairwise interaction elements are obtained by summing up the contributions of interaction elements from each possible interaction between P1P2, P1P3, P1P4, P1P5, P2P3, P2P4, P2P5, P3P4, P3P5, and P4P5 respectively. Mathematically, the total number of interaction elements is $(2 \times 2) + (2 \times 3) + (2 \times 3) + (2 \times 2) + (2 \times 3) + (2 \times 3) + (3 \times 3) + (3 \times 2) + (3 \times 2) = 57$.

Table 2. All 2-Way Possible Interaction Elements for the Pizza System

P1P2	P1P3	P1P4	P1P5	P2P3	P2P4	P2P5	P3P4	P3P5	P4P5
VC,TC	VC,RC	VC,La	VC,EI	TC,RC	TC,La	TC,EI	RC,La	RC,EI	La,EI
VC,ET	VC,GB	VC,Me	VC,TA	TC,GB	TC,Me	TC,TA	RC,Me	RC,TA	La,TA
ML,TC	VC,Mu	VC,Sm	ML,EI	TC,Mu	TC,Sm	ET,EI	RC,Sm	GB,EI	Me,EI
ML,ET	ML,RC	ML,La	ML,TA	ET,RC	ET,La	ET,TA	GB,La	GB,TA	Me,TA
	ML,GB	ML,Me		ET,GB	ET,Me		GB,Me	Mu,EI	Sm,EI
	ML,Mu	ML,Sm		ET,Mu	ET,Sm		GB,Sm	Mu,TA	Sm,TA
							Mu,La		
							Mu,Me		
							Mu,Sm		

To demonstrate the interaction element coverage analysis for $t = 2$, this work summons the authors' implementation of HSS [5, 6] for generating the test suite for the aforementioned pizza system as in Table 3. In this case, the HSS produces 9 test cases for the pizza system.

Here, a number of variables for their interaction coverage metric per test case are defined. Firstly, weight is defined as the number of covered interaction by a particular test case. In this case, the maximum weight for a particular test case is 10 there are 10 possible two way interactions between parameters. Cumulative weight signifies the total number of covered interaction elements from the first test case to the current test case of interests. For our pizza system example, it is expected that the cumulative weight for the last test case is 57, which is equal to the total number interaction elements. As the name suggests, uncovered interaction UI denotes the number of uncovered interaction. The uncovered ratio UCR is defined as the ratio of UI over the total number of interaction or simply:

$$UCR = \frac{\text{Uncovered interactions}}{\text{Total number of interaction}} \quad (1)$$

Table 3. A Possible 2-way Test Suite Covering All Interactions for Pizza System

No	P ₁ P ₂ P ₃ P ₄ P ₅	Interaction covered	Cumulative Weight	UI	UCR
1	VC,ET,RC,La,TA	{VC,ET},{VC,RC},{VC,La},{VC,TA},{ET,RC},{ET,La},{ET,TA},{RC,La},{RC,TA},{La,TA}	10	47	0.824
2	ML,TC,RC,Me,EI	{ML,TC},{ML,RC},{ML,Me},{ML,EI},{TC,RC},{TC,Me},{TC,EI},{RC,Me},{RC,EI},{Me,EI}	20	37	0.65
3	VC,ET,Mu,Sm,EI	<u>{VC,ET}</u> , {VC,Mu},{VC,Sm},{VC,EI},{ET,Mu},{ET,Sm},{ET,EI},{Mu,Sm},{Mu,EI},{Sm,EI}	29	28	0.49
4	ML,TC,GB,Sm,TA	<u>{ML,TC}</u> , {ML,GB},{ML,Sm},{ML,TA},{TC,GB},{TC,Sm},{TC,TA},{GB,Sm},{GB,TA},{Sm,TA}	38	19	0.333
5	VC,ET,GB,Me,EI	<u>{VC,ET}</u> , {VC,GB},{VC,Me}, <u>{VC,EI}</u> , {ET,GB},{ET,Me}, <u>{ET,EI}</u> , {GB,Me},{GB,EI}, <u>{Me,EI}</u>	44	13	0.28
6	ML,TC,Mu,La,EI	<u>{ML,TC}</u> , {ML,Mu},{ML,La}, <u>{ML,EI}</u> , {TC,Mu},{TC,La}, <u>{TC,EI}</u> , {Mu,La}, <u>{Mu,EI}</u> , {La,EI}	50	7	0.123
7	VC,TC,Mu,Me,TA	{VC,TC}, <u>{VC,Mu}</u> , <u>{VC,Me}</u> , <u>{VC,TA}</u> , <u>{TC,Mu}</u> , <u>{TC,Sm}</u> , <u>{TC,TA}</u> , {Mu,Me},{Mu,TA},{Me,TA}	54	3	0.053
8	ML,ET,RC,Sm,EI	{ML,ET}, <u>{ML,RC}</u> , <u>{ML,Sm}</u> , <u>{ML,EI}</u> , <u>{ET,RC}</u> , <u>{ET,Sm}</u> , <u>{ET,EI}</u> , {RC,Sm}, <u>{RC,EI}</u> , <u>{Sm,EI}</u>	56	1	0.018
9	ML,TC,GB,La,EI	<u>{ML,TC}</u> , <u>{ML,GB}</u> , <u>{ML,La}</u> , <u>{ML,EI}</u> , <u>{TC,GB}</u> , <u>{TC,La}</u> , <u>{TC,EI}</u> , {GB,La}, <u>{GB,EI}</u> , <u>{La,EI}</u>	57	0	0

Referring to Table 3, this paper denotes the interaction elements that are covered multiple times (redundant) using underline and bold fonts. Initially, this study notes that the first two test cases obtain the maximum possible cumulative weight of 20. In the third test case, the interaction element consisting of {VC,ET} is redundant as it has already been covered in first test case. For this reason, the cumulative weight for the third test case is 29. Similar observation can be seen for the fourth test case. From the fifth test cases onward (i.e. until the final test case), there exist multiple redundant interaction elements that have already been covered by earlier test cases. Also, in the final test case, this paper notes that only one new interaction element is covered to complete all 57 interaction elements.

Concerning UI and UCR, their respective values gradually change with each test case. It is expected that UI and UCR value are going on a downward trend. Apart from its direct relationship with UCR as referred to equation 1, UI value is also related to cumulative weight. For the aforementioned pizza example, at any test case, the sum of UI and cumulative weight must always equal to the total number of interaction elements. For this reason, the last value for UI and UCR is 0.

To highlight the analysis in different perspective, the XY graph taking UCR as the x axis and test case number as the y axis is plotted as in Fig. 2. From the graph, this study concludes that 100% coverage of all the two way interactions could be achieved by only 9 test cases out of the exhaustive 72 test cases.

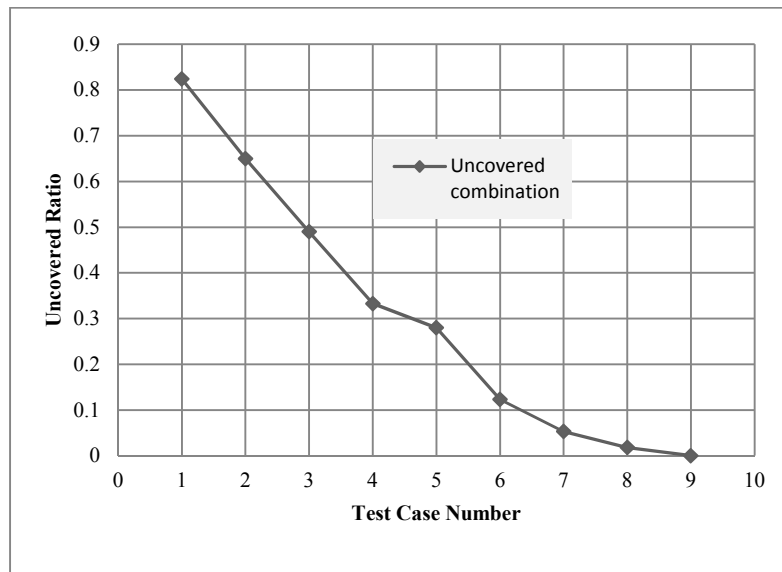


Fig. 2. Uncovered Ratio versus Test Case Number

4.0 EXPERIMENTAL RESULTS AND DISCUSSION

Using the same analysis discussed earlier, this section analyzes and benchmarks existing AI-based strategies in terms of their interaction coverage metric per test case. The strategies of interests are Hill Climbing HC [9], Simulated Annealing SA [2], Tabu Search TS [10], Great Flood GF, Particle Swarm Test Generator PSTG [3-5], and Harmony Search Algorithm [6, 7]. Two configurations based on thirteen 3-valued parameters and mixed parameters, for example ten 2-valued parameters, three 3-valued parameters, two 4-valued parameters, and one 5-valued parameter, have been selected for analysis, each of which with the interaction strength t of 4.

The results for HC, SA, TS, GF, and PSTG are taken from [11] and [12], whereas the authors use their own HSS strategy in order to get the result for HSS. This study running environment consists of a desktop PC with Windows XP, 2.8 GHz Core 2 Duo CPU, and 2 GB of RAM. Noted here is the fact that there is also a need for comparing against the greedy algorithm, because all the AI-based strategies are used with the greedy algorithm.

The experiments for HC, SA, TS, GF, and PSO are run five times each and the average of the runs is reported [11]. Hence, this paper considers the same experiment condition for HSS. In addition, the experiment undertaken in [11] used 1000 repetitions for each algorithm in order to get the best results. However, HSS achieve the same result merely with 5 repetitions [7]. Tables 4 and 5 represent the analysis of results whilst Fig. 3 and 4 depicts the XY plot of UCR against test case number.

Referring to Table 4, HSS appears to be the greediest strategy initially with low UCR metric ratio as compared to other strategies in the first 50 test cases. Nonetheless, towards the end, PSTG steadily outperforms HSS. The whole interactions are covered by 252 test cases for HSS, while they are covered by more test cases with the other AI-based strategies. Putting HSS and PSTG aside, SA outperforms the rest of the strategies with low overall UCR metric. The result for TS and GF appears to be sufficiently competitive. Nonetheless, the Greedy algorithm and HC give poor performance overall. In fact, the greedy algorithm and HC has large uncovered interaction UI in the last test case 6 and 12 respectively. In this case, it may be that greedy algorithm and HC require more than 350 test cases to cover all the interactions, thus, indicating poor test generation performance in terms of test size optimality.

In Table 5, both HSS and PSTG initially give the best overall performance with low UCR metrics. Nonetheless, towards the end HSS manage to outperform PSTG, as seen in the table HSS requires 352 test cases to cover all interactions. Apart from HSS and PSTG, GF and SA also yield low UCR metric for early part of the test cases. Nonetheless, unlike GF, the performance of SA deteriorates significantly towards the end. Greedy algorithm and HS give the poorest performance overall with the last UCR metric value of 0.001 and 0.0003 respectively.

Table 4. The 4-way Interaction Elements Coverage Metric for the Input 3^{13}

Test No.	Greedy		HC		SA		TS		GF		PSTG		HSS	
	Repetition No.		Repetition No.		Repetition No.		Repetition No.		Repetition No.		Repetition No.		Repetition No.	
	0		1,000		1,000		1,000		1,000		20		5	
	UI	UCR	UI	UCR	UI	UCR	UI	UCR	UI	UCR	UI	UCR	UI	UCR
25	41,692	0.719	41,629	0.719	40,719	0.703	40,947	0.707	40,845	0.705	40,811	0.705	40,689	0.703
50	29,365	0.507	29,460	0.509	26,895	0.464	27,359	0.472	27,007	0.466	26,878	0.464	26,837	0.463
75	20,497	0.354	20,056	0.346	16,989	0.293	17,634	0.305	16,952	0.293	16,799	0.290	16,811	0.290
100	14,039	0.242	13,868	0.239	10,165	0.176	10,986	0.189	10,187	0.176	9,933	0.172	9,942	0.172
125	9,383	0.162	9,364	0.162	5,641	0.097	6,595	0.114	5,724	0.099	5,538	0.096	5,584	0.096
150	6,271	0.108	6,267	0.108	2,976	0.051	3,795	0.066	3,002	0.052	2,825	0.049	2,951	0.051
175	4,154	0.071	4,101	0.070	1,453	0.025	2,075	0.036	1,488	0.026	1,327	0.023	1,432	0.024
225	1,545	0.026	1,592	0.027	180	0.003	492	0.008	214	0.004	155	0.003	174	0.003
250	916	0.015	932	0.016	10	0.0001	188	0.003	13	0.0002	4	6E-05	8	0.0001
275	484	0.008	523	0.009	0	0	32	0.0005	0	0	0	0	0	0
300	201	0.003	252	0.004	0	0	0	0	0	0	0	0	0	0
325	59	0.001	81	0.001	0	0	0	0	0	0	0	0	0	0
350	6	0.0001	12	0.0002	0	0	0	0	0	0	0	0	0	0

Table 5. The 4-way Interaction Elements Coverage Metric for the Input $2^{10}3^34^25^1$

Test No.	Greedy		HC		SA		TS		GF		PSTG		HSS	
	Repetition No.		Repetition No.		Repetition No.		Repetition No.		Repetition No.		Repetition No.		Repetition No.	
	0		1,000		1,000		1,000		1,000		20		5	
	UI	UCR	UI	UCR	UI	UCR	UI	UCR	UI	UCR	UI	UCR	UI	UCR
25	47,357	0.575	45,010	0.547	44,420	0.539	45,049	0.547	44,422	0.539	44,267	0.538	44,311	0.538
50	29,766	0.362	25,764	0.313	24,625	0.299	25,580	0.310	24,491	0.298	24,445	0.297	24,481	0.297
75	19,724	0.239	15,485	0.188	14,438	0.175	15,318	0.186	14,340	0.174	14,225	0.173	14,279	0.174
100	13,143	0.159	9,496	0.115	8,522	0.104	9,425	0.115	8,493	0.103	8,484	0.103	8,506	0.103
150	6,051	0.074	3,822	0.045	3,029	0.037	3,734	0.045	3,079	0.037	2,969	0.036	3,014	0.037
200	2,786	0.034	1,510	0.018	1,065	0.013	1,480	0.018	1,059	0.013	1,043	0.013	1,071	0.013
250	1,146	0.014	566	0.007	345	0.004	543	0.007	333	0.004	330	0.004	344	0.004
300	376	0.005	177	0.002	101	0.001	151	0.002	93	0.001	91	0.001	90	0.001
350	96	0.001	28	0.0003	16	0.0002	11	0.0001	8	9E-05	4	4E-05	2	2E-05

Based on the UCR versus test case number plot in Figs. 3, and 4, this study notes that there is no significant difference in term of the “greedy” performance between all AI-based strategies as the lines overlap with each other. This is expected as AI-based strategies are normally good optimizers. Nonetheless, the greedy algorithm appears to be out-of-the league as far as the resulting test cases are concerned.

Finally, this paper notes that the results achieved by HC, SA, TS and GF are with 1000 iterations and PSO with 20 iterations. HSS requires only 5 iterations to achieve the aforementioned results suggesting the superiority of Harmony search algorithm for t-way testing applications.

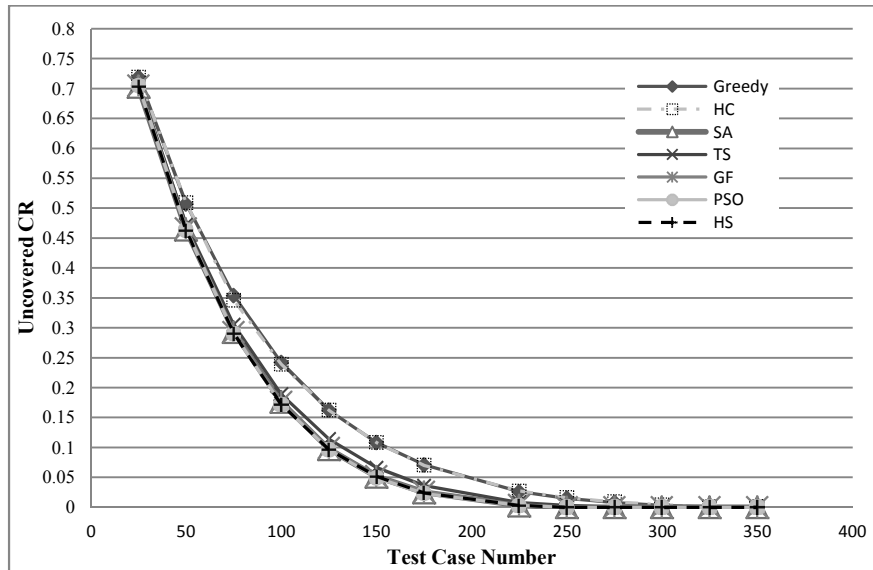


Fig. 3. UCR versus Test Case Number for 3^{13}

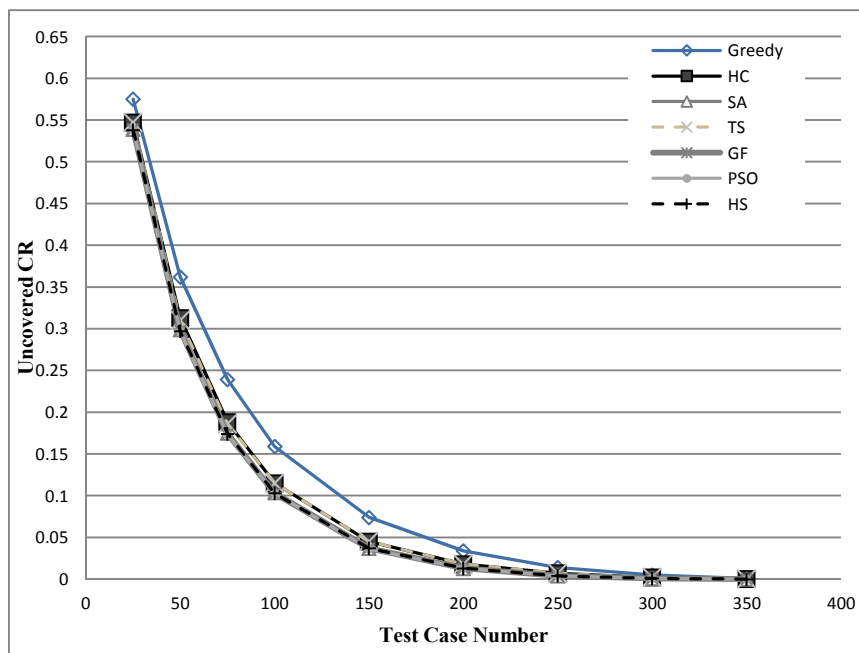


Fig. 4. UCR versus Test Case Number for $2^{10} 3^3 4^2 5^1$

4.0 CONCLUSION

In conclusion, this paper has highlighted a systematic analysis of existing AI-based strategy using interaction coverage metric per test case to indicate the measure of greediness a particular strategy of interest is. In doing so, the paper also highlights the potential of analyzing the test suite in order to help tester to choose and prioritize the best test cases with high coverage ratio. As the scope for future work, this research is currently investigating an interaction coverage tool that can gradually report the interaction coverage automatically given any test suite outputs.

ACKNOWLEDGEMENT

This research is partially funded by the generous Short Term Grant (“Development of a Pairwise Interaction Testing Strategy with Check-pointing Recovery Support”) from Universiti Malaysia Pahang.

REFERENCES

- [1] K. Z. Zamli, M. F. J. Klaib, M. I. Younis, N. A. M. Isa, and R. Abdullah,” Design And Implementation Of A T-Way Test Data Generation Strategy With Automated Execution Tool Support” Information Sciences, Vol. 181, No. 9, 2011, pp. 1741-1758.
- [2] M. B. Cohen, “Designing Test Suites for Software Interaction Testing”, in Department of Computer Science, University of Auckland: New Zealand 2004.
- [3] B. S. Ahmed and K. Z. Zamli, “ PSTG: A T-Way Strategy Adopting Particle Swarm Optimization”, in Proceedings of 4th Asia International Conference on Mathematical /Analytical Modelling and Computer Simulation: IEEE Computer Society 2010.
- [4] B. S. Ahmed and K. Z. Zamli, and C. P. Lim, “ Constructing a T-Way Interaction Test Suite Using the Particle Swarm Optimization Approach”, International Journal of Innovative Computing, Information and Control, Vol. 8, No. 1, 2012, pp. 431-452.
- [5] B. S. Ahmed and K. Z. Zamli, “A Variable-Strength Interaction Test Suites Generation Strategy Using Particle Swarm Optimization”, Journal of Systems and Software, Vol. 84, No. 12, 2011, pp. 2171-2185.
- [6] A. R. A. Alsewari and K. Z. Zamli, “ Interaction Test Data Generation Using Harmony Search Algorithm”, in Proceeding of IEEE Symposium on Industrial Electronics & Applications. Langkawi, Malaysia: IEEE Computer Society 2011.
- [7] A. R. A. Alsewari and K. Z. Zamli, “ Design and Implementation of a Harmony-search-based Variable-strength T-Way Testing Strategy with Constraints Support”, Information and Software Technology, Vol 54, No. 6, 2012, pp. 553-568.
- [8] A. R. A. Alsewari and K. Z. Zamli, “ A Harmony Search Based Pairwise Sampling Strategy for Combinatorial Testing. International Journal of the Physical Sciences, Vol. 7, No. 7, 2012, pp. 1062 1072.
- [9] C. Nie and H. Leung, ” A Survey of Combinatorial Testing”, ACM Computing Surveys (CSUR), Vol 43, No. 2, 2011, pp. 1-11.
- [10] K. J. Nurmela, “ Upper Bounds for Covering Arrays by Tabu Search”, Discrete Applied Mathematics, Vol. 138, No. 1-2, 2004, pp. 143-152.
- [11] R. Bryce and C. Colbourn, “ One-Test-at-a-Time Heuristic Search for Interaction Test Suites” in Proceedings of the 9th annual conference on Genetic and evolutionary computation. London, England: ACM 2007.
- [12] B. S. Ahmed and K. Z. Zamli, “ Comparison of Metahuristic Test Generation Strategies Based on Interaction Elements Coverage Criterion”, in Proceeding of IEEE Symposium on Industrial Electronics and Applications (ISIEA) 2011.
- [13] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence,” IPOG/IPOG-D: Efficient Test Generation for Multi-way Combinatorial Testing”, Software Testing Verification and Reliability, Vol. 18, No. 3, 2008, pp. 125-148.

- [14] Y. Lei and K. C. Tai, “ In-Parameter-Order: A Test Generation Strategy for Pairwise Testing”, in Proceedings of The 3rd IEEE International Symposium on High-Assurance Systems Engineering. Washington, DC, USA 1998.
- [15] Lei. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, “ IPOG: A General Strategy for T-Way Software Testing” in Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems. Tucson, AZ U.S.A 2007.
- [16] NIST. ACTS 2010 [cited 2010 6 June]; Available from: <http://csrc.nist.gov/acts>
- [17] M. I. Younis, K. Z. Zamli, and N. A. M. Isa, “ MIPOG-Modification of the IPOG Strategy for T-Way Software Testing” in Proceeding of the Distributed Frameworks and Applications (DFmA) 2008.
- [18] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton,” The AETG System: An Approach to Testing Based on Combinatorial Design”, IEEE Transactions on Software Engineering, Vol 23, No. 7, 1997, pp. 437-444.
- [19] A.W. Williams,” Determination of Test Configurations for Pair-wise Interaction Coverage” in Proceeding of the 13th International Conference on Testing of Communicating Systems: Kluwer, B.V. 2000.
- [20] P.J. Schroeder, E. Kim, J. Arshem, and P. Bolaki,” Combining Behavior and Data Modeling in Automated Test Case Generation”, in Proceedings of the 3rd International Conference on Quality Software: IEEE Computer Society 2003.
- [21] T. Yu-Wen and W. S. Aldiwan,” Automating Test Case Generation for the New Generation Mission Software System” in Proceedings of the IEEE Aerospace Conference. Big Sky, MT, USA: IEEE Computer Society 2000.
- [22] Pallas, D.: Jenny. Available from: <http://www.burtleburtle.net/bob/math> 2003.
- [23] M. Grindal, J. Offutt, and S. F. Andler,” Combination testing strategies: a survey” Software Testing, Verification & Reliability, Vol 15, No. 3, 2005, pp. 167-199.
- [24] W. Afzal, R. Torkar, and R. Feldt,” A Systematic Review of Search-Based Testing for Non-Functional System Properties”, Information and Software Technology, Vol. 51, No. 6, 2009, pp. 957-976.
- [25] C. J. Colbourn and J. H. Dinitz,” Handbook of Combinatorial Designs”, ed. S. Edition. Chapman & Hall/CRC 2006.

BIOGRAPHY

AbdulRahman A. Alsewari obtained his BEng in Computer Engineering from the Military Engineering College, Baghdad in 2002, his MEng in Electronic System Design Engineering from the Universiti Sains Malaysia in 2009, and his PhD in Software Engineering from Universiti Sains Malaysia in 2012. His research interests include Software Engineering and Software Testing.

Norazlina Khamis is currently attached to Department of Software Engineering, Universiti of Malaya since year 2000. She received her MSc in real-time software engineering from Universiti Teknologi Malaysia in 2001, and PhD in Software Engineering from Universiti Kebangsaan Malaysia in 2012. Her main research interests are in software quality, software engineering education and educational technology.

Kamal Z. Zamli obtained his BSc in electrical engineering from Worcester Polytechnic Institute, USA, in 1992; his MSc degree in real-time software engineering from Universiti Teknologi Malaysia in 2000; and his PhD in software engineering from University of Newcastle upon Tyne, UK, in 2003. Professor Dr Kamal is currently attached to the Faculty of Computer Systems and Software Engineering, Universiti Malaysia Pahang. His research interests include software engineering, t-way testing, and algorithm design.