Find the Best Greedy Algorithm with Base Choice Experiments for Covering Array Generation

Jing Jiang and Changhai Nie

The State Key Laboratory for Novel Software Technique, Nanjing University, Nanjing, China

Abstract. A number of greedy algorithms have been conducted for covering array construction, and most of them can be integrated into a framework, and more approaches can be derived from the framework. However, such a framework is affected by many factors, which makes its deployment and optimization very challenging. In order to identify the best configuration, we design Base Choice experiments based on six decisions of the framework to study systematically, providing theoretical and practical guideline for the design and optimization of the greedy algorithms.

1 Introduction

Many modern systems are built by components; unexpected interactions among components may cause some potential system failure. Combinatorial testing has been proposed as a means to detect failures triggered by the interactions among components in Software Under Test (SUT)[2].

For instance, an Internet-based software system in which end-users may use a variety of web browsers, operating systems, connection types and memory configurations, as shown in Table 1. To exhaustively test all possible combinations needs $3^4 = 81$ test cases. In this system each component is a *factor*, and each setting of the component is a *level* for the factor. As the combinatorial explosion of larger systems prohibits exhaustive testing, it is a challenge to detect failures caused by interactions among the different factors.

Combinatorial testing has been proposed as a means to offer significant savings. We reduce to 9 test cases by employing pair-wise interaction testing (shown in Table 2). All individual pairs are tested instead of testing every combination. Given larger system with ten factors each having four levels, we only need 25 test cases by employing pair-wise interaction testing, instead of $4^{10} = 1,048,576$ test cases by exhaustive testing

Table 1. The online system.

Web Browser	Operating System	Connection Type	Memory
Netscape	Windows	LAN	256MB
IE	Macintosh	PPP	512MB
Mozilla	Linux	ISDN	1GB

Jiang J. and Nie C..
Find the Best Greedy Algorithm with Base Choice Experiments for Covering Array Generation.
DOI: 10.5220/0003559200530060
In *Proceeding of the 1st International Workshop on Evidential Assessment of Software Technologies* (EAST-2011), pages 53-60
ISBN: 978-989-8425-58-4

Table 2. The online system.

Test *No.*	Web Browser	Operating System	Connection Type	Memory
1	Netscape	Windows	LAN	256MB
2	IE	Macintosh	LAN	512MB
3	Netscape	Macintosh	PPP	1GB
4	Mozilla	Linux	LAN	1GB
5	Netscape	Linux	ISDN	512MB
6	IE	Linux	PPP	256MB
7	Mozilla	Windows	PPP	512MB
8	IE	Windows	ISDN	1GB
9	Mozilla	Macintosh	ISDN	256MB

[10]. This sampling approach is scientific and effective. Kuhn et al. examined fault reports for several systems. They showed that more than 70% of defects can be caught with 2-way interactions [3].Overall consideration of the cost of testing, the execution time to generate covering arrays and the array size, pair-wise testing is considered as a practical method.

In order to generate combinatorial test suites, a combinatorial object called covering array is often used. Covering array $MCA(N; t, k, (v_1, v_2, \cdots, v_k)$ is an $N \times k$ array on v symbols, where $v = \sum_{i=1}^{k} v_i$, t is the *strength* of the coverage of interactions, and k is the number of factors. Each column i $(1 \leq i \leq k)$ contains only elements from a set V_i with $|V_i| = v_i$. The rows of each $N \times t$ sub-array cover all t-tuples of levels from the columns at least once [2]. Table 2 is a 2-way covering array in which all combinations between every two factors are covered by the nine test cases.

Covering array generation is a key issue in combinatorial testing. Early combinatorial methods can provide fast generation. However, they depend on the existence of specific algebraic of combinatorial objects. For example TConfig [4] is a recursive construction method based on orthogonal arrays, and can generate covering array effectively, but it depends on the existence of the corresponding orthogonal array. In terms of heuristic search, Simulated Annealing [5] and Tabu Search [6] can produce many small covering arrays, but it is very time-consuming. In addition to heuristic search methods, there are many greedy methods such as AETG [7, 8], TCG [9], DDA [10], IPO [11] and so on. We have also proposed some greedy algorithms to generated test suites [13, 14].

In this paper, we focus on a series of existing greedy methods including AETG [7], TCG [9] and DDA [10]. Bryce [1] has integrated these methods into a unified framework, these methods can not only be included by this framework, but also more new approaches can be derived from it. However, such a framework is affected by multiple factors, which makes its deployment and optimization very challenging. Bryce et al. used ANOVA to analysis the effect of each decision in array size[1], but they did not give any concrete usable configuration to create effective greedy algorithm from the framework. We design and conduct Base Choice[12] experiments under the framework with six decisions. Through the experiments we can find the best configuration to build the most effective greedy algorithm from the framework. It provides practical guideline for the design and optimization of the greedy algorithms.

The remainder of this paper is organized as follows: Section 2 briefly introduces the greedy algorithm framework, Section 3 describes the Base Choice experiment design, Section 4 and 5 analyze the experimental data, and Section 6 presents a summary and the future work.

2 Framework of the Greedy Algorithms

Bryce [1] proposed a four layer framework with six decisions from AETG,TCG and DDA. Fig. 1 provides the detail of the greedy framework. Six decisions need to be made (see Table 3), shown in shadows in the skeleton of Fig. 1. For convenience, we write six decisions as $f_0, f_1, \cdots, f_5$.We will explain them one by one next.

1. Select a factor f_i according to the factor ordering selection criterion.
2. In the case of more factors selected, then choose the factor f_i by factor tie-breaking.
3. Assign a level $l_i$for the factor f_i according to the level selection criterion.
4. In the case of more levels selected, then choose the level l_i by level tie-breaking.
5. Repeat the process until all factors have been fixed, then create a test case.
6. Repeat the above steps Candidates times, candidate rows are generated.
7. Choose a candidate that covers the most new pairs into the covering array.
8. Repeat the above steps until all pairs have been covered, the covering array is complete.
9. Repeat the above steps Repetitions times, then choose a smallest covering array.

Fig. 1. The framework of greedy algorithm.

- **Decision One – Repetitions (f_0):** Due to the randomness of certain decisions, smaller covering arrays may be generated by repetitions [1]. In this paper, we only consider four kinds of repetitions: *1*, *5*, *10* and *20* repetitions.
- **Decision Two – Candidates (f_1)**: The algorithm may generate a number of rows as candidates, and choose the one adding the most new pairs into the covering array [1]. Here we set the numbers of candidates as *1*, *5*, *10* and *20*.
- **Decision Three – factor Ordering (f_2)**: The factor ordering is the essence of the framework. Researchers have refined five strategies [1]: (1) *uncovered pairs*, by the number of new pairs involving the factor and the fixed factors; (2) *density*, by expected number of pairs covered involving both fixed and free factors; (3) *level*, by the number of associated levels; (4) *random*; (5) *hybrid*, the first factor is selected by *uncovered pairs*, and remaining factors are ordered randomly.
- **Decision Four – Level Selection(f_3):** Its goal is to cover the largest number of new pairs. Bryce has raised three criterions [1]: (1) *random*; (2) *uncovered pairs*, by the number of new pairs involving the level of the current factor and the fixed factors; (3) *density*, by the expected number of new pairs associated with fixed and free factors.
- **Decision Five – Factor Tie-breaking (f_4):** When employing the strategy of factor ordering, the algorithm may suffer from ties. To break ties, one of the following methods may be used [1]: *take first*, *random*, *uncovered pairs*.

Table 3. The specific strategies of six decisions.

No.	f_0	f_1	f_2	f_3	f_4	f_5
0	1	1	random	random	random	random
1	5	5	uncovered pairs	uncovered pairs	uncovered pairs	uncovered pairs
2	10	10	density	density	take first	take first
3	20	20	level			least used
4			hybrid			

– **Decision Six – Level Tie-breaking(f_5):** The level tie-breaking is also needed, the following methods are used [1]: *random*, *take first*, *uncovered pairs* and *least used*.

3 Experimental design

In this paper, we compare the performance of the greedy algorithms on the size of the generated covering array, and aim to answer the following questions: (1) does the configuration of the framework affect the size of the generated covering array? (2) If the configuration of the framework has an impact on the performance of the generated covering array size, can we find an optimal configuration to generate smaller covering arrays for some systems? (3) If the optimal configuration exists in some specific systems, is it able to construct smaller covering arrays for the other systems? (4) Is the optimal configuration of the framework competitive with AETG, TCG and DDA?

To address these questions, we employ a sampling method –Base Choice to study all kinds of methods based on the framework. We produce a configuration set with Base Choice method, this method starts by identifying a base configuration, subsequent configurations are constructed by varying the choices of one decision at a time and keeping the choices of the other decisions fixed on the base configuration [12]. The process is not finished until the configuration set covers all choices for the six decisions. For example, we select base configuration randomly as B1 = (2, 2, 1, 1, 2, 0), Table 4 is a configuration set generated by Base Choice strategy. (We use natural numbers (*No.*) to denote each choice of the decisions in Table 3, B1 = (2, 2, 1, 1, 2, 0) represents the configuration (10, 10, uncovered pairs, uncovered pairs, take first, random).)

4 Experimental Analysis

Using the Base Choice configuration set (Table 4), we configure the framework and get 18 greedy algorithms, then generate covering arrays with them for five systems listed in the first row of Table 5, and record the *size* of the generated covering array respectively in Table 5. For example, for the system $3^4 4^5$ (which means a system with 9 factors, 4 factor having 3 levels, 5 factors having 4 levels), we can get 24 test cases with configuration B1. The letter *"f"* denotes that the greedy method is failed to generate a covering array.The results in Table 5 demonstrate that the configurations of the framework have a significant impact on the performance of the generated covering array size. The experimental results are analyzed in the following.

Table 4. Base Choice configuration set.

No.	f_0	f_1	f_2	f_3	f_4	f_5	*No.*	f_0	f_1	f_2	f_3	f_4	f_5
B1	2	2	1	1	2	0	B10	2	2	3	1	2	0
B2	0	2	1	1	2	0	B11	2	2	4	1	2	0
B3	1	2	1	1	2	0	B12	2	2	1	0	2	0
B4	3	2	1	1	2	0	B13	2	2	1	2	2	0
B5	2	0	1	1	2	0	B14	2	2	1	1	0	0
B6	2	1	1	1	2	0	B15	2	2	1	1	1	0
B7	2	3	1	1	2	0	B16	2	2	1	1	2	1
B8	2	2	0	1	2	0	B17	2	2	1	1	2	2
B9	2	2	2	1	2	0	B18	2	2	1	1	2	3

Table 5. Results for Base Choice experiments.

	$5^1 3^8 2^2$	$3^4 4^5$	9^4	6^4	$8^6 7^5$		$5^1 3^8 2^2$	$3^4 4^5$	9^4	6^4	$8^6 7^5$
B1	20	24	94	42	97	B10	20	23	97	43	102
B2	21	26	94	42	97	B11	20	25	93	41	101
B3	20	24	93	42	97	B12	29	33	132	55	173
B4	19	23	92	41	97	B13	20	24	95	43	97
B5	21	25	102	44	106	B14	20	23	92	42	97
B6	20	24	93	43	99	B15	20	23	93	43	97
B7	20	24	93	42	95	B16	20	25	100	44	103
B8	21	25	94	41	101	B17	*f*	*f*	*f*	*f*	*f*
B9	20	22	93	41	96	B18	21	*f*	*f*	44	*f*

- **Repetitions (f_0):** The configurations B1, B2, B3 and B4 only vary the number of *repetitions* in B1. The numbers of *repetitions* are studied using settings of *10*, *1*, *5* and *20*. From Table 5, we can find that by increasing the number of repetitions, the array size can reduce slightly, it improves the size performance at the cost of time. For example, with configuration B2, the greedy algorithm can generate a covering array of 26 test cases in .437s for $3^4 4^5$, with B4, the size is 23 in 8.25s. As the tradeoff between the time and the generated covering array size, $f_0 = 20$ is the best choice.
- **Candidates (f_1):** The configurations B1, B5, B6 and B7 just change the number of *candidates*. Its settings are *10*, *1*, *5* and *20* respectively. The results in Table 5 indicate that more candidates can reduce the array size. But when the candidates are increased to some extent, the performance tends toward stability. Moreover the increase of candidates wastes time. Also as the trade-off between the size and time cost, $f_1 = 20$ is the best choice.
- **Factor Ordering (f_2):** The difference of the configurations B1, B8, B9, B10 and B11 is only the choice of *factor ordering*. Their choices are *uncovered pairs*, *random*, *density*, *level* and *hybrid* respectively. *Density* in B9 appears to be the best choice, and *uncovered pairs* factor ordering is competitive. Only *random* factor ordering produces the worst results. The performance of other two choices is the average. So $f_2 = density$ is the best choice.

- **Level Selection(f_3):** There are three *level selection* methods under the configurations B1, B12 and B13: *uncovered pairs*, *random* and *density*. From Table 3, B12 with *random* level selection always generate the largest covering array. Both *density* in B13 and *uncovered pairs* in B1 have good performance. However *uncovered pairs* level selection is a little better than *density*, so we let f_3=*uncovered pairs* be the best choice.
- **Factor Tie-breaking (f_4):** The choices of the *factor tie-breaking* in the configurations B1, B14 and B15 are *take first*, $random$ and *uncovered pairs* respectively. From Table 5, the three choices have similar performance, so we let f_4=*take first*, *random* or *uncovered pairs*.
- **Level Tie-breaking(f_5):** The choices of the *level tie-breaking* in configurations B1, B16, B17 and B18 are *random*, *uncovered pairs*, *take first* and *least used*. *Take first* yields very poor performance, frequently suffering from a dead loop. In addition the methods that use *least used* would fail occasionally. We can fine the optimal choice for level tie-breaking is f_5=$random$.

Based on the above analysis, *20 repetitions*, *20 candidates*, *density* factor ordering, *uncovered pairs* based on level selection, free choices for factor tie-breaking and *random* level tie-breaking can be the optimal choice for each decision, totally we can obtain three best configurations $Best1 = (3, 3, 2, 1, 0, 0)$,$Best2 = (3, 3, 2, 1, 1, 0)$ and $Best3 = (3, 3, 2, 1, 2, 0)$.

Table 6. Comparison with published results.

	Best1	Best2	Best3	DDA	AETG	TCG
3^{13}	18	18	18	18	15	20
$5^1 3^8 2^2$	19	20	19	21	19	20
$6^1 5^1 4^6 3^8 2^3$	33	33	34	34	34	33
$5^1 4^4 3^{11} 2^5$	26	26	26	27	30	30
$4^{15} 3^{17} 2^{29}$	34	34	34	35	41	35
$7^1 6^1 5^1 4^5 3^8 2^3$	42	42	42	43	45	45
4^{40}	43	44	44	43	42	46

5 Verifying Experiments

To verify the optimal configurations Best1, Best2 and Best3 of the above experiments, we make some verifying experiments, and examine the following two aspects: 1) are the optimal configurations able to generate smaller covering arrays for the other systems; 2) are they competitive with the existing methods AETG, TCG and DDA?

For evaluation, we generate covering arrays with them for the seven systems in the first column of Table 6, and the experiments confirm that the optimal configurations also do well in generating covering array for the systems . In addition, we compare the published results for AETG, TCG and DDA in the literature [7, 9, 10]. From Table 6, we can see that the configuration Best1 is a little better than other two configurations, so we select Best1 as our optimal configuration. We can find that the optimal configuration Best1 is competitive in generating covering arrays. For example, for the system

$5^1 4^4 3^{11} 2^5$, the size of the generated covering array by Best1 is 26, the size by AETG is 30, the size by TCG is 30, and the size by DDA is 27.

6 Conclusions

We studied a greedy framework with six decisions built by Bryce [1]. Thousands of greedy methods can be derived from this framework. In order to find the best algorithm, we employ Base Choice method [12] to systematically sample an amount of greedy algorithms derived from the framework. According to the experimental results, we can draw the following conclusions: (1) the configurations of the framework have a significant impact on the performance of the covering array size; (2) We can obtain an optimal configuration in some specific systems, and (3) the optimal configuration can work for the other systems as well; (4) Comparing the optimal configuration with the existing methods AETG, TCG and DDA, we find that the optimal configuration has its advantages, it can generate smaller covering array than the existing methods.

Our conclusion is a complementary and verification to Bryce's results. We find more repetitions and candidates may decrease the covering array size, but it requires more time cost. Moreover, while these two factors are increased to some extent, the size no longer decreases. We also find the *random* factor ordering yields very poor performance.

In the future work, we plan to conduct more profound and comprehensive studies on the greedy framework, which may include: (1) consider more choices of the framework; (2) employ other more scientific sampling methods to optimize the framework; (3) consider the cases of seeds and constraints in covering array generation.

Acknowledgements

This work was supported by the National Natural Science Foundation of Jiangsu province (BK2010372), the National Natural Science Foundation of China (60773104,60721002), 863 high technical plan of China (2009AA01Z143).

References

1. R. C. Bryce, C. J. Colbourn, M. B. Cohen: A Framework of Greedy Methods for Constructing Interaction Test Suite. In: Proceedings of 27th international conference on software engineering (ICSE2005). St. Louis, Missouri, USA, May 15-21, 2005:146–155.
2. Changhai Nie, Hareton Leung: A suvery of combinatorial testing. ACM Computing Survey, 2011, 43(2).
3. D. Kuhn and M. Reilly. An investigation of the applicability of design of experiments to software testing. Proc. 27th Annual NASA Goddard/IEEE Software Engineering Workshop, October 2002.
4. A. W. Williams, R. L. Prober. A Practical Strategy for Testing Pair-wise Coverage of Network Interfaces. In Proceedings of 7th International Symposium on Software Reliability Engineering (ISSRE1996), White Plaints, NY, USA, October 30-November 2, 1997: 246–254.
5. M. B. Cohen, P. B. Gibbons, W. B. Mugridge, C. J. Colbourn. Constructing Test Suites for Interaction Testing. In Proceedings of the 25th International Conference on Software Engineering (ICSE2003), Portland, Oregon, USA, May 3–10, 2003: 38–48.

6. K. J. Nurmela. Upper Bounds for Covering Arrays by Tabu Search. Discrete Applied Mathematics, 2004, 138(1–2): 143–152.
7. D. M. Cohen, S. R. Dalal, M. L. Fredman, G. C. Patton: The AETG system: an approach to testing based on combinatorial design. IEEE Transactions on Software Engineering, 23(7):437–44, October 1997.
8. D. M. Cohen, S. R. Dalal, M. L. Fredman, J. Parelius, and G. C. Patton. The combinatorial design approach to automatic test generation. IEEE Software, 13(5):82–88, October 1996.
9. Y. Tung, W. Aldiwan: Automating test case generation for the new generation mission software system. IEEE Aerospace Conf. , pages 431–37,2000.
10. R. C. Bryce, C. J. Colbourn: The density algorithm for pairwise interaction testing. Journal of Software Testing, Verification, and Reliability, 2007.
11. K. C. Tai, Y. Lei. A Test Generation Strategy for Pairwise Testing. IEEE Transaction on Software Engineering, 2002, 28(1): 109–111.
12. M. Grindal, B. Lindastrom, A. J. Offutt, S. F. Andler: An Evaluation of COmbination Strategies for Test Case Selection. Empirical Software Engineering, 2006,11:583–611.
13. C. H. Nie, B. W. Xu, Z. Y. Wang, L. Shi. Generating Optimal Test Set for Neighbor Factors Combinatorial Testing. QSIC 2006: 259–265.
14. C. H. Nie, B. W. Xu, L. Shi, Z. Y. Wang. A new Heuristic for Test Suite GEneration for Pair-wise Testing. SEKE 2006:517–521.