Using computational search to generate 2-way covering array

Changhai Nie and Baowen Xu
State Key Laboratory for Novel Software Technology
Nanjing University, Nanjing 210093, China
Hareton Leung
Hong Kong Polytechnic University, China

April 28, 2009

Software and its running environments are becoming more and more complex, and there are more and more factors that can influence the behavior of the software. For a software under test SUT, suppose there are n factors which may affect its execution, each factor(parameter) c_i has a_i ($1 \leq i \leq n$)values, and the value set is denoted as V_i, we need $a_1 \times a_2 \times \cdots a_n$ test cases if we do exhaustive testing. When n and a_i are large, test cost will increase quickly. From observation, we need not run all the test cases, as sometimes there is only a small number of parameters which can trigger failure. So we just need to design a test suite to cover all the combinations of some parameters. Covering array CA is a $m \times n$ matrix, and each row represents a test case for SUT, with all the values in the ith column from V_i, the value set of parameter i. If for every τ parameters of SUT: $c_{i_1}, c_{i_2}, \cdots, c_{i_\tau}$, each value combination in $V_{i_1} \times V_{i_2}, \cdots, V_{i_\tau}$ can be found in some row of CA, CA is called τ-way covering array. When τ is not fixed in CA, it is called variable strength covering array(VCA). When $\tau = 2$, it is 2-way covering array. In this paper we focus on generating 2-way covering array.

Much work has been done to generate covering array. Computational search methods, including Heuristic search techniques (Hill Climbing(HC), Great Flood(GF), Tabu Search(TS), Simulated Annealing(SA)) and AI-based search techniques(Genetic Algorithm (GA), Ant Colony Algorithm(ACA)), have been applied to τ-way covering array and VCA generation. These methods usually start from a pre-existing test set and then apply a series of transformations to the test set until it covers all the combinations [1]. [2] used GA-based technique that identifies a set of test configurations that are expected to maximize pairwise coverage with a predefined number of test cases. [3] augmented the one-test-at-a-time greedy algorithm with a heuristic search. [4] reported using the computational method of simulated annealing to generate 3-way covering array and variable strength array. [5] also used GA and ACA to generate 3-way covering array. Computational search techniques can often produce a smaller test set than those from the greedy algorithm, but typically require a longer computation. In the following we give a generic computational search algorithm for covering array generation.

Framework of computational search algorithm
Let **S** *be the set of all the τ-way combinations that should be covered according to the model of SUT. Let* **Seeds** *be the set of test cases assigned by testers, remove all the τ-way combinations covered by the test cases in* **Seeds** *from* **S**.
For each test case t, fitness(t)=the number of uncovered τ-way combinations.
*While (***S**$\neq \emptyset$*)*
Generate a set of test cases randomly;
Evolve the test set with any search techniques, such as SA, HC, PSO, ACA, GA, etc.

Choose the best t with the highest fitness.
Remove all the τ-way combinations covered by the test cases t from **S**.
End while.

We reimplemented GA, PSO (Particle Swarm Optimization), ACA, SA, HC and developed a random algorithm (RA) for 2-way covering array generation. Figure 1 gives the size of test suite generated by these different search techniques. In the first column, v^n means that SUT has n parameters and each parameter has v values.

Random Algorithm
Input n*: the number of parameters;*
a_i*: the number of values of parameter* $i(1 \leq i \leq n)$
Output: 2-way covering array
Let **S** *be the set of all pairs that need to be covered;*
while $(\mathbf{S} \neq \emptyset)$
$bestf = 0$; //fitness of the current best test case;
bestt: the initial best test case;
for $j = 1$ to 1000(or 10000 ,can be adjusted)
Randomly generate a test case t;
$fitness(t)$ =number of uncovered pairs in t;
If $fitness(t) = n(n-1)/2$ output t and delete all the pairs covered by t from **S** and break
Else if $(bestf < fitness(t))$ *bestt*=t and $bestf = fitness(t)$;
End for;
Output *bestt* and delete all the pairs covered by t from **S**;
End while;

The five search techniques all do well in the 2-way covering array generation, but none of them can outperform others. The random method is also not better than any of these five algorithms, although sometimes it can yield a smaller size covering array faster. The random method is easier and quicker.

For future work we will explore more search techniques,such as TS,GF,etc, to generate covering array,including τ-way covering array($\tau > 2$) and VCA, do more experiments to compare their performances and to build a smaller covering array.

v^n	GA	PSO	ACA	SA	HC	RA
3^4	10	13	14	9	10	10
3^{13}	20	22	21	20	19	20
2^{100}	16	16	18	16	16	15
10^{20}	195	301	265	279	178	295
4^{10}	29	31	32	32	31	31
4^{20}	37	45	44	44	44	45
4^{50}	45	66	64	62	62	64
6^{10}	63	70	72	70	69	72
6^{20}	76	101	97	101	100	101
8^{20}	128	183	174	178	178	186
8^{40}	161	256	233	252	242	249
10^{30}	225	359	322	337	338	350

Figure 1: Comparison of test suite size generated by different search techniques

References

[1] Harman, M.. The current state and future of search based software engineering. In Future of Software Engineering, FOSE 07, 342-357.

[2] Ghazi, S.A. Ahmed, M.. Pair-wise test coverage using genetic algorithms. In Proc. the 2003 Congress on Evolutionary Computation. IEEE Computer Society,Washington, DC, USA, 1420-1424.

[3] Bryce, R. C. and Colbourn, C. J.. One-test-at-a-time heuristic search for interaction test suites. In Proc. the 9th annual conference on Genetic and evolutionary computation. ACM, New York,USA, 1082-1089.

[4] Cohen, M. B., Gibbons, P. B., Mugridge, W. B., and Colbourn, C. J.. Constructing test suites for interaction testing. In Proc. the 25th International Conference on Software Engineering. Washington, DC, USA, 38-48.

[5] Shiba, T., Tsuchiya, T., and Kikuno, T.. Using artificial life techniques to generate test cases for combinatorial testing. In Proc. the 28th Annual Inter- national Computer Software and Applications Conference. Washington, DC, USA, 72-77.