International Journal of Control Theory and Applications

ISSN : 0974-5572

Volume 10 • Number 36 • 2017

An Analysis of Broad Constraint Command (BCC) Test Strategy

S. Bhuvana[a] and M.V. Srinath[b]

[a]*Research Scholar, Department of Computer Science, S.T.E.T Womens College, Mannargudi, Thiruvarur (D.t.).*
[b]*Ph.D. Research Advisor & Director, Department of Master of Computer Application, S.T.E.T Womens College, Mannargudi, Thiruvarur (D.t.).*

Abstract: Broad Constraint Command (BCC) is a new strategy capable of supporting high interaction strength. In this domain the complete input interface test is not a realistic solution to test all software configurations for generating the test cases. A previous investigation many software testing problems involve series of procedures, in the Combinatorial Interaction Testing (CIT) is enhanced by only the minimal test suite for covering the test cases, and that the same way Late Acceptance Based Hill Climbing Algorithm(LAHC) is a t-way test strategy, it will also support only the minimum amount of parameters are used to constraints the software product line testing and so that the next level of test cases are configured by using the Enhanced Combinatorial Interaction Testing Software (ECITS) algorithm. The Enhanced Combinatorial Interaction Testing Software (ECITS) algorithm is enhanced by three –way testing provided the better results than existing two – way testing techniques and there was no statistically significant change among the methods for three-way testing ,but n-way combinatorial test was detected the number of random tests. Considering the high interaction strength is not in the absence of complications. The entire work will be integrated and the time duration of work is shorter than other software development models. But testing occupies the major part of the software development- the maximum time of the allotted time is utilized the testing in BCC also. The major objective of this work is test cases in Broad Constraint Command (BCC) based model. Broad Constraint Command (BCC) adopts the straight and cross extensions in order to construct the desired test data. This paper describes about the test cases based on Broad Constraint Command (BCC) values are used perform the analysis to load the test cases from given software and generate the test cases as early as possible and it will consume minimum amount time of time generate the test cases and that the same time the test case amount is also very less.

Keywords: Broad Constraint Command (BCC), Enhanced Combinatorial Interaction Testing Software (ECITS), Combinatorial Interaction Testing (CIT), Combinatorial Testing (CT), Test Data (TD), Constraint Set (CS).

1. INTRODUCTION

Testing is an essential but costly quantity of the software growth process. Deficiency of testing frequently pointers to terrible moments as well as damage of records, wealth and even is alive. For these details, a lot of input constraints and scheme settings necessity to be tested along side the terms of the scheme for conformance. Even if looked-for, extensive testing is too expensive even in an adequate-sized assignment, owing to the properties as

fine as skill constraints[2]. Consequently, it is required to reduce the test collection galaxy in an organized method. In track with aggregate user difficulties for original functionalities and inventions, software implementations developed vast in extent above the past 15 years. This unexpected development takes a reflective control as far as testing is troubled. Now, the test sort developed expressively as an end. In the direction of the statement the above-mentioned problems, much investigation is currently aiming on sampling methods based on interaction testing (termed n-way testing strategy) in command to arise the most optimal test suites for testing reflection (i.e., termed as Test Suite (TS) for even constraint values and Mixed Test Suite (MTS) for uneven constraint values in turn). Before implementation of n-way testing provided mixed outcomes. Even though 3-way testing (also named Enhanced Combinatorial Interaction Testing Software (ECITS)) performs to be sufficient for completing fine test reporting in the existing method, an opposite argument recommends that such a decision cannot be universal to all (upcoming) software system. Frequently, the remaining result of software development presents new interlace addition among constraints associated with, accordingly, justifying the necessity to support for high relations power (t).

Previous work on three-way testing has principally absorbed on pair wise testing, which purposes to identify errors that are initiated by relations among any two constraints. Still, errors can also be initiated by relations, including more than two constraints. In direction to excellently identify those errors, it is needed to allow an advanced power of reporting. In this paper, we simplify a new strategy, called Broad Constraints Command (BCC) for n-way testing. A main test of our simplification, energy is allocated with the combinatorial development in the size of combinations of constraint-values. We refer to an n-way testing, and argue strategy results that are complete to assist a well-organized application of the Broad Constraints Command (BCC) test strategy. We also report some testing that were shown to assess the value of n-way. In certain, we showed a testing that compared n-way to existing test strategy. The end of this test shows that n-way achieved expressively improved than the other test strategy for a real-life application.

The remaining segments of this paper are prepared as follows. Segment 2 argues some related work. Section 3 gives the details of Enhanced Combinatorial Interaction Testing Software(ECITS) and our modified Broad Constraints Command (BCC). The similarities and differences between the two are also explained. Section 4 highlights comparisons between BCC and ECITS test method. Finally, Section 5 gives the conclusions.

2. RELATED WORK

T-Way test generation strategy based on Late Acceptance Based Hill Climbing Algorithm (LAHC): The authors described in this paper, the *t*-way strategy based on the Late Acceptance Based Hill Climbing Algorithm(LAHC). Late Acceptance Based Hill Climbing Algorithm (LAHC) generated a current neighbor to be compared with all the corresponding value of the Late Acceptance Based Hill Climbing Algorithm (LAHC) memory one-at-a-time. It also maintains the previous cost function in the memory to allow selection of the best fit value. The *t*-way results have been promising as Late Acceptance Based Hill Climbing Algorithm (LAHC) gives competitive results in most constraint configurations considered[2]. And these methods future work is to extend the capability of (LAHC) in terms of supporting high constraints to be used for software product line testing.

T-way test generation using Bees Algorithm (BA): In this research, the authors highly recommend for implementation of Bees Algorithm(BA) for generating test cases to detect *t*-way interaction faults. The *t*-way strategy, in this paper divided in two main parts. First, the test data generated and stored using Bees algorithm (BA). And the stored test data generates all possible interactions for the intended constraints. Second, the Bees algorithm (BA) optimized the intended constraints test data's. The applied algorithm running every time the number of interaction set will be reduced[6]. The Bees algorithm (BA) keeps running until the interaction set is

empty. In future the applied algorithm of this paper also supports variable strength interaction while generating *t*-way test data.

Combinatorial Testing (CT) Method: Event Sequence Testing: In this sequence testing, applied Combinatorial Testing (CT) to testing difficulties that has 1, 2, …, *n* distinct events, where every event occurs exactly once. The sequence Covering Array (CA), as the name suggests are similar to standard Covering Array (CA), which contain at smallest one of every *t*-way combination of any n variables, where $t < n$. A variability of processes are obtainable for creating Covering Array (CA) but there are not usable for producing *t*-way sequences as they are planned to shield combinations in any order. Using a sequence Covering Array (CA) for scheme testing defined here made it promising to arrange for better assurance that the system would function correctly regardless of possible dependencies among peripherals[8]. This technique ensures that any testing events will be tested in every possible *t*-way order.

3. PROPOSED METHOD

As argued previously, the proposed strategy, BCC is based on the existing Enhanced Combinatorial Interaction Testing Software (ECITS) 3-way strategy. For a system with at least n or more constraints, the Broad Constraints Command (BCC) strategy makes an n-way test data formation of the first n constraints. Then, it covers the test data to build an n-way test data of n+1constraints. After that, it continues to extend the test data until an n-way test data has been constructed for all the constraints of the system. Like Broad Constraints Command (BCC) performs the horizontal growth followed by the vertical growth, but in a different way in order to optimize the number of generating test sizes such that the n-way interaction element is covered by the minimum number of test cases.

As the inputs to Broad Constraints Command (BCC) algorithm is the degree of interaction 'n' and the set of constraints 'cs'. The output is an n-way test data of all the constraints in the system. The differences between the two strategies lie in both horizontal and vertical extensions.

Broad Constraints Command (BCC): Algorithm

BCC-Test (int n, ConstraintSet 'cs')

{

Step 1: initialize test data *td* to be an empty set

Step 2: denote the constraints in *cs*, in an arbitrary order, as C_1, C_2, …, and C_t

Step 3: add into test data *td* a test for each combination of values of the first *n* constraints

Step 4: for (int $i = n + 1$; $i \leq t$; i++){

Step 5: let π be the set of *n*-way combinations of values involving constraints C_i and $n - 1$ constraint between the first $i - 1$ constraints

Step 6: for (each test $\tau = (v_1, v_2, \ldots, v_{i-1})$ in testdata td)

{

Step 7: if (τ not contains don'tmind)

{

// don't mind means that there is a previous constraint(s) that not assigned value (s). As such, it can be further optimized choose a value v_i of C_i and replace τ with $\tau' = (v_1, v_2, \ldots, v_{i-1}, v_i)$ so that τ' covers the maximum number of combinations of values in π}

Step 8: else

{

choose a value v_i of C_i and search all possible tuples that can be optimized the don't mind to construct $\tau' = (v_1, v_2, \ldots, v_{i-1}, v_i)$ so that τ' covers the maximum number of combinations of values in π and optimized the don't mind

}

Step 9: remove from π the combinations of values covered by τ'}

Step 10: while (π not empty){

Step 11: rearrange π in decreasing order according to the size of the remaining tuples

Step 12: Choose the first tuple and generate testcase (τ)

that combine maximum number of tuples

Step 13: delete the tuples covered by τ, add τ tolocal td

Step 14: } //while

Step 15: return td;}

In the straight extension, the Broad Constraints Command (BCC) strategy checks all the values of the input constraints, and chooses the value that contains the maximum number of combinations for the uncovered tuples in the π set. BCC also optimizes the 'don't care' value. For this reason, Broad Constraints Command (BCC) always generates a stable test case (that cannot be changed) by searching for tuples that can be covered by the same test. This is performed by means of searching of uncovered tuples that can be combined with the test case to fill the 'don't care' values during the straight extension (i.e. to ensure that the test case is indeed optimized). In the cross extension, BCC rearranges the π set in a decremented order size. After that, Broad Constraints Command (BCC) chooses the first tuple from the rearranged π set and combines the tuple with other suitable tuples in the π set (i.e. the resulting test case must have the maximum weight of uncovered tuples). Once combined, all the tuples are removed from the π set. This process is repeated until the π set is empty (i.e. to ensure the complete interaction coverage). τ means test services that evaluates to a given input values and that the Test Data(TD) values are generalized using the conditions. Constraint Set (CS) included the arbitrary order of set $C_1, C_2, \ldots, C_t$. It will be satisfies the condition to the given Test Data (TD).

4. RESULT & DISCUSSION

In this section, we evaluate Broad Constraints Command (BCC) with the following objectives:

(i) To investigate the overall performance of Broad Constraints Command(BCC).

(ii) To investigate whether Broad Constraints Command (BCC) can have a significant gain against ECITS in terms of test size result.

In the (t) denotes → Interaction Strength

In the (c) denotes → No. of Constraints

In the (v) denotes → No. of values

From Table 1, we can see that Broad Constraints

Command (BCC) yields different execution time for 2 to 7 and fix the number of constraint (C) to 10 and different size of codes with various values 'V' from strength of coverage (t) to 5.

Table 1
Results for Broad Constraints Command (BCC) with Value (V) = (2…7)

S.No.	*No. of Value (V)*	*BCC*	
		LOC	*TIME*
1	2	97	0.181
2	3	657	0.623
3	4	3057	5.298
4	5	8577	48.25
5	6	24735	804.91
6	7	52077	1433.72

We plot values against size (Lines of Code), Which shows size is proportional quinary (that is minimum of five values) with the number of values.

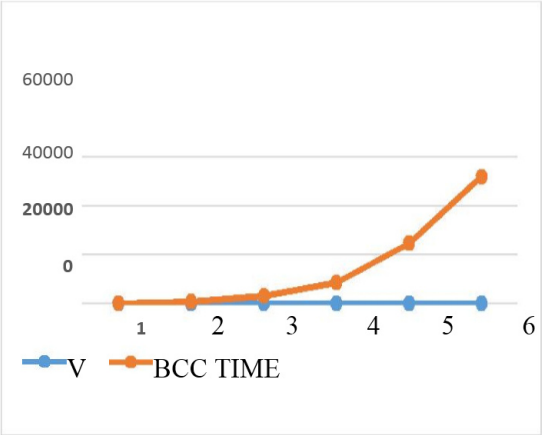

Figure 1: Results for *n*-way with Broad Constraint Command (BCC) on the basis of the Constraints

Figure 1 represents that *n*-way with Broad Constraints Command (BCC) on the basis of the Value (V).

The graph shows, the maximum number of values is 6.

From Table 2, we can see that Broad Constraints Command (BCC) yields different execution time for different size of codes with various constraints ranging from 7 to 13 and fix both values and strength of coverage to 5.

Table 2
Results for Broad Constraints Command (BCC) with Constraints (C) = (7, …, 13)

SL.NO	*Constraints (C)*	*BCC*	
		LOC	*TIME*
1	7	3281	1.02
2	8	5906	3.03
3	9	7346	13.197
4	10	8578	25.685
5	11	9520	60.316
6	12	10473	154.862
7	13	11554	347.277

We plot constraints against execution time, which shows execution time grows in quinary (that is minimum of five constraints) with respect to the logarithmic scale of constraints.

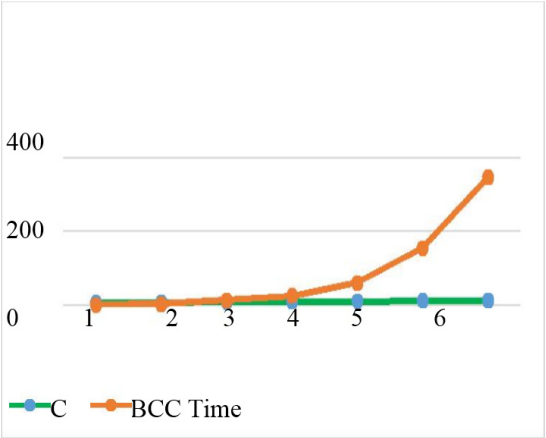

Figure 2: Results for *n*-way with Broad Constraints Command (BCC) on the basis of the Constraints

The *n*-way with Broad Constraints Command (BCC) on the basis of the Constraints (C) is shown in Figure 2. The graph shows, the maximum number of constraints is 6.

From Table 3, We can see that Broad Constraints Command (BCC) yields different execution time for different size of codes with various interaction strength '*t*' from 2 to 7 and fix the constraints 'C' to10 and value to 10.

Table 3
Results for Broad Constraints Command (BCC) with Time (T) = (2, 3, …, 7)

S.No.	*Time (T)*	*BCC*	
		LOC	*TIME*
1	2	66	0.35
2	3	537	1.887
3	4	3839	72.50
4	5	24735	804.91
5	6	14662	13901.92
6	7	736405	48572.85

We plot interaction strength against size, which shows test size grows exponentially as the strength of coverage increases.

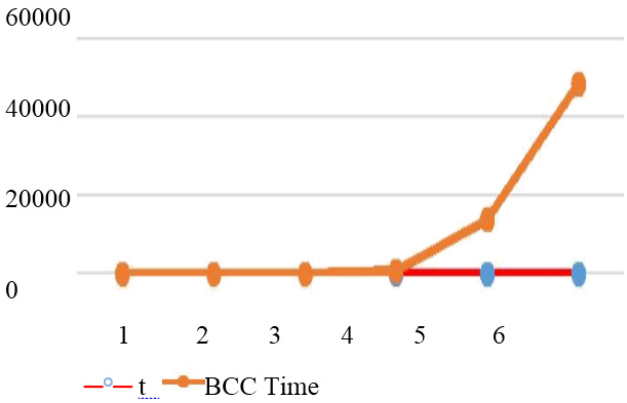

Figure 3: Results for *n*-way with Broad Constraints Command (BCC) on the basis of the Strength

The Broad Constraints Command (BCC) on the basis of the Time (T) is represents the *n*-way test case in Figure 3. The graph shows ,the maximum number of Time is 6.

5. RESULT OF TESTING EFFICIENCY

This section describes the performance analysis of the Broad Constraints Command (BCC) approach for the following metric.The implementation of the Broad Constraints Command (BCC) is done in the platform of visual studio 2008 and MYSQL5.6.17. The sample screens.

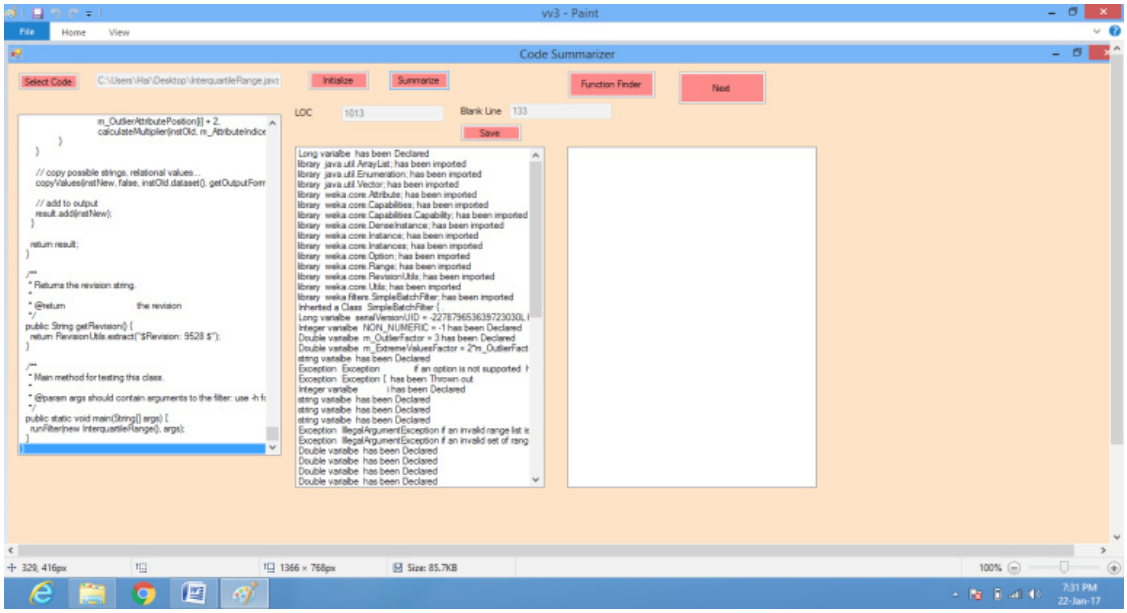

Figure 4: Summarization screen shot

The summarization screen shots are represent Figure 4, that is explanation of given software coding, but this concept already discussed in the Enhanced Combinatorial Interaction Testing Software (ECITS) algorithm.

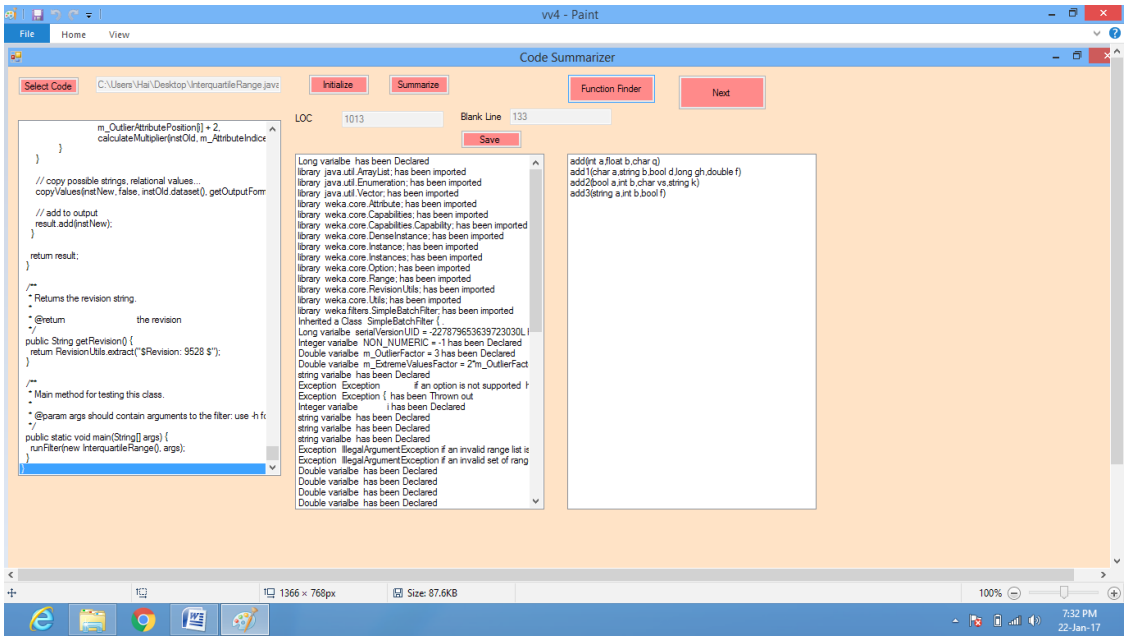

Figure 5: Function Finder

Figure 5 shows that all the functions used in that software that will be separtated. Because these functions used to generate the test cases.

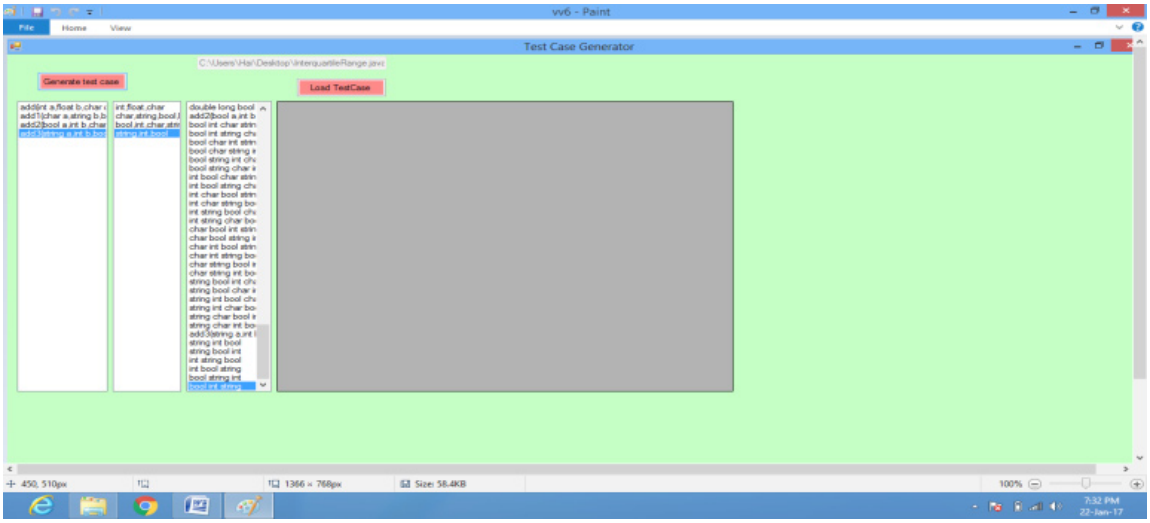

Figure 6: Generate the Test cases

Figure 6 shows that the test cases are generated, these test cases are updated by using the given functions. That functions are generated by the using the given software.

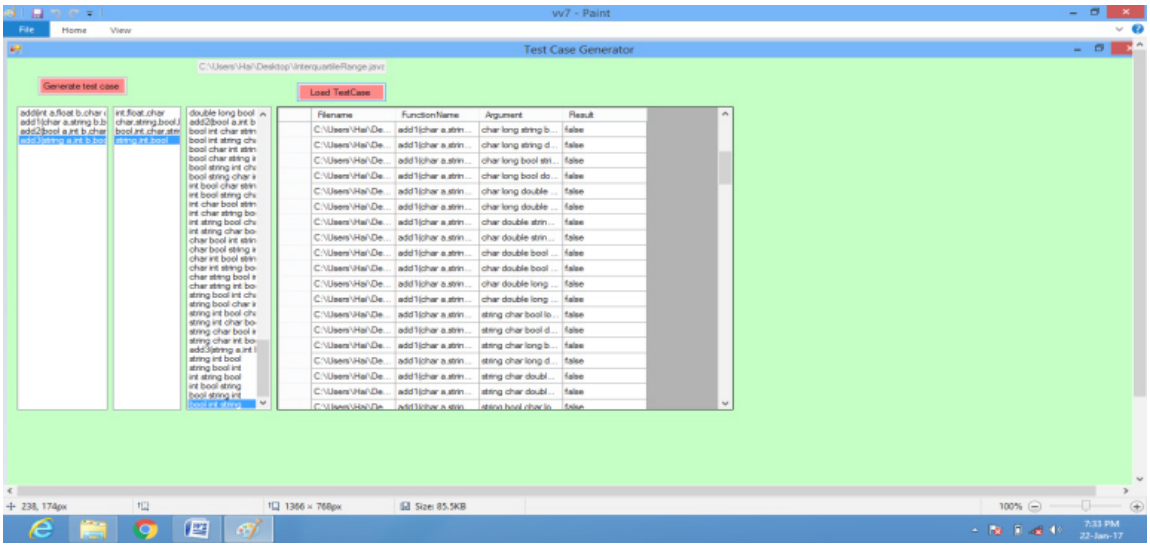

Figure 7: Sample screens of proposed work Broad Constraints Command (BCC)

The proposed work of Broad Constraints Command (BCC) will be generated in Figure 7, that is test cases are generated first and then these test cases are loaded.

6. PERFORMANCE ANALYSIS

Broad Constraints Command (BCC) is perhaps the most developed form of Enhanced Combinatorial Interaction Testing Software (ECITS). Testers have used it for years with two-way, three-way, four-way but it is a n-way coverage. Many if not most software systems have a large number of configuration parameters. Many of the earliest applications of combinatorial testing were in testing all pairs of system configuration.

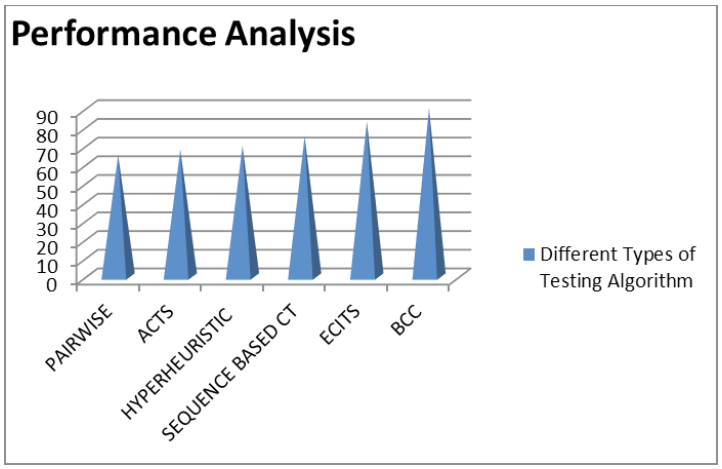

Figure 8: Performance Analysis

Because the number of tests required grows only logarithmically with the number of events.

But using a BCC it is *n*-way system testing described here made it of extensive human involvement, the time required for a single test is significant, and a small number of random tests or scenario-based ad hoc testing would be unlikely to provide *t*-way sequence coverage to a satisfactory degree possible to provide greater confidence that the system would function correctly regardless of possible dependencies among peripherals. Because of extensive human involvement, the time required for a single test is significant, and a small number of random tests or three-way, four-way would be unlikely to provide n-way testing coverage to a satisfactory degree.

7. CONCLUSION

In this paper, we have proposed an efficient n-way test data generator Broad Constraints Command (BCC) based on constraints. BCC adopts the straight and cross extensions in order to create the desired test data, and that the same time it generates the small test data sizes with a standard performance time, our estimation of BCC is promising. In fact, our experience also indicates BCC is capable to generate a higher strength test suite that has never been described in the collected works (i.e., $t > 6$). We're currently working with developers of real-world software to measure the costs and benefits of this approach for full-scale systems. We are integrating BCC within the web background in order to attain more speed as far as performance time is troubled. In addition, we will expand the strategy to handle other practical testing issues such as needs among features and ethics.

REFERENCES

[1] R.Kuhn, Y.Lei & R.Kacker, Practical Combinatorial Testing: Beyond Pairwise, in IT Professional, Vol: 10, Issue: 3, May 2008.

[2] Kamal Z. Zamli, Abdul Rahman Alsewari & Basem Al-Kazemi, "Comparative Benchmarking of Constraints T-Way Test Generation Strategy Based on Late Acceptance Hill Climbing Algorithm", in IJSECS, Vol:1, PP: 15-27, February 2015.

[3] Renee C. Bryce, SreedeviSampath & Atif M. Memon, "Developing a Single Model and Test Prioritization Strategies for Event-Driven Software", in Software Engineering, Vol: 37, Issue: 1, PP: 48 – 64, February 2011.

[4] C.Yilmaz, E.Dumlu, Myra B. Cohen & A.Porter, "Reducing Masking Effects in Combinatorial Interaction Testing: A Feedback Driven Adaptive Approach", in Software Engineering, Vol: 40, Issue: PP: 43 – 66, January 2014.

[5] M.Rahman, R.R.Othman, & RB.Ahmad, "Test case generation for event driven systems using 4-way input teststrategy", Research and Development (SCOReD), December 2015.

[6] M.Hazli, M.Zabil & K.Z. Zamli "Implementing a T-Way Test Generation Strategy Using Bees Algorithm", in IJASCA, Vol: 5, Issue: 3, December 2013.

[7] C.Yilmaz, "Test Case-Aware Combinatorial Interaction Testing", in Software Engineering, Vol: 39, Issue:5, PP: 684 – 706, May 2013.

[8] D. Richard Kuhn, James M. Higdon, James F. Lawrence & Raghu N. Kacker, Y.Lei, "Combinatorial Methods for Event Sequence Testing", April 2012.

[9] Christopher Henard, Mike Papadakis & Gilles Perrouin, "Bypassing the Combinatorial Explosion : Using Similarity to Generate and Prioritize T-Wise Test Configurations for Software Product Lines", in IEEE Software Engineering, Vol-40, Issue-7, PP-650 –670, 2014.

[10] Justyna Petke, Myra B. Cohen, Mark Harman & Shin Yoo, "Practical Combinatorial Interaction Testing: Empirical Findings on Efficiency and EarlyFault Detection", in IEEE Software Engineering, Vol-41, Issue-9, PP-901 – 924, 2015.

[11] Mohammad F. J. Klaiba, Mohammad Subhi Al-batahb & Rashad J. Rasrasc, "3-way Interaction Testing using the Tree Strategy", ICCMIT 2015,Procedia Computer Science65(2015)845 – 852.

[12] Cemal Yilmaz, "Test Case-Aware Combinatorial Interaction Testing", in IEEE Software Engineering, Vol-39, Issue-5, PP-684 – 706, 2013.

[13] M.Zamri Zahir Ahmad, R. Razif Othman & M.S.Aziz Rashid Ali, "Sequence Covering Array Generator (SCAT) For Sequence Based Combinatorial Testing", in IJAER, Volume11, Issue:8, PP:5984-5991, 2016.

[14] Ely Porat & Amir Rothschild, "Explicit Nonadaptive Combinatorial Group Testing Schemes", in IEEE Information Theory, Vol-57, Issue-12, PP-7982 – 7989, 2011.

[15] M.N. Borazjany, L.Yu & Y.Lei, "Combinatorial Testing of ACTS: A Case Study", 2012 IEEE Fifth International Conference on Software Testing.