Green-JEXJ: A new tool to measure energy consumption of improved concolic testing

Sangharatna Godboley*, Arpita Dutta†, Bhagyashree Besra ‡, and Durga Prasad Mohapatra§
DOS Lab, Department of Computer Science and Engineering,
National Institute of Technology Rourkela, Rourkela, Odisha, India.*†‡§
Email: sanghu1790@gmail.com*,arpitad10j@gmail.com†,bubblingkajal@gmail.com‡,and durga@nitrkl.ac.in§

***Abstract*—Green Computing is a solution for environment sustainability, where any part of computer does not pollute the environment system. The main focus of Green Computing is to reduce harmful material used in computers, boost the energy efficiency and create recyclability of waste. Now a days energy consumption of software is very interesting research area. Green Software Testing deals with applying Green Computing technique on software testing. In this paper, we developed a tool to measure energy consumption of improved concolic testing. We have named our tool as Green-JEXJ (Green JEXNCT JCUTE). We have developed Green-JEXJ in Java language with a nice Graphical User Interface (GUI). The GUI provides 100% augmented testing technique on Green-JEXJ. Our proposed method achieves 7.45% of average increase in branch coverage for some sample programs. The average difference of energy consumption of the sample programs is -75945.1008 Joules ('-ve' shows that our transformation approach is energy efficient).**

I. Introduction

Green Computing is a technique of using systems and technology in echo-friendly manner. It is also known as Green technology. This concept uses computer and related resources in environmentally responsible manner. This technique involves using the energy efficient central processing unit, servers, and peripherals. This also involves the proper disposal of E-waste material [11].

The SMART 2020 report reveals that carbon dioxide emissions from the ICT sector will represent an estimate of 2.8% of the total global emission by the year 2020 [7]. Some of the impacts of ICT on human health and environment are the followings: hazardous electronics waste, health risk, climate change, global warming and land and water pollution etc. Fig. 1 depicts the components of GreenICT implementation explored from literature review which focuses on green ICT implementation practices, factors, benefits, and barriers to the educational institutions [7]. Ministry of Environment and Forests Government of India (2011) have developed National Mission for Green India under the National Action Plan on Climate Change (NAPCC). The has presented tentative action plan for Implementation of the Green India Mission during 2011-2012.

Software testing is an important phase in Software Development Life Cycle (SDLC). Earlier days it was a manual practice, but currently it is a must to automate software testing to save the efforts in terms of time and cost. Software testing is of two types: Black-box testing and White-box testing. Coverage based testing is a white-box testing technique that

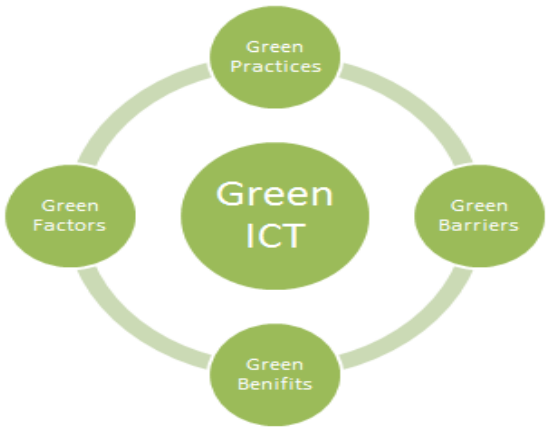

Fig. 1. Components of Green ICT Implementation

depends on structure of the code. In Nuclear and Aerospace critical safety systems, branch coverage is mandatory. Now a days Concolic Testing is very much useful for software testing practices because it does exhaustive and rigorous testing. We propose a technique to improve concolic testing. Since, energy consumption, branch coverage, and concolic testing are important in safety critical systems. So, we apply green computing technique on coverage based testing and concolic testing. By this hybrid nature of the work, we deal with Green Software Testing. Being a software testing engineer, it is our responsibility to spread awareness on Green Software Testing. We should measure how much power and energy are consumed for our testing tools or testing technique. It is our responsibility to develop such testing tools, so that they generate energy efficient test cases. There are lot of ideas pending to apply Green Computing on Software testing technique.

In our proposed work, we develop a tool named Green JEXNCT JCUTE (Green-JEXJ) that measures Energy consumption of our proposed testing technique. Green-JEXJ consists of five modules JEXNCT, JCUTE, Speed Calculator, JouleMeter, and Energy Consumption Calculator. Our proposed tool spreads the awareness of energy consumption of an improved concolic testing.

The rest of the paper is organized as follows: Section 2 discusses some of the fundamental ideas required to understand the proposed approach. Section 3 explains the proposed

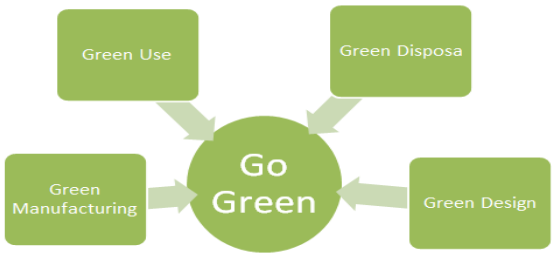

Fig. 2. Green computing concept

Green-JEXJ tool. In Section 4, we present the experimental result. In Section 5, we compare our work with some of the existing related work. Section 6 concludes with some insights into our future work.

II. Fundamental Ideas

In this section, we discuss some basic concepts and important terminologies that we use in our proposed approach. Green Computing is a major concept which can be able to change the whole digital life. There are many tools, plans, and techniques present for reducing the power consumption but recycling and reusability are major issues that can be solved with the awareness of end users and with the intelligence of developers. To reduce the consumption of power, virtualization, power management techniques, and cooling systems are used. Many tips, plan, tools and technologies are used for the purpose of Green Computing [9]. The Green Computing concept is shown in Fig. 2.

Definition 1: Green Use: It is a concept, which reduces the power consumption of computer system and use them in environmentally sound manner.

Definition 2: Green Disposal: It deals with recycle and reuse of computer waste.

Definition 3: Green Design: It deals with reducing hazardous material production.

Definition 4: Green Manufacturing: It deals with manufacturing computers and other sub parts with minimal impact on environment.

It is very important to understand some terms on the path to the evolution of Green Software. These terms are defined below:

Definition 5: Green IT: Green IT is the study and practice of designing, manufacturing, using and disposing IT related hardware products in an efficient sustainable way with minimal or even no impact on the environment [14].

Definition 6: Green Software Engineering: Green Software Engineering is the attempt to apply "green principles" known from hardware products, also on software products, software development processes and their underlying software process models [14].

Definition 7: Sustainable Software: Sustainable Software is the software whose direct and indirect negative impacts on economy, society, human beings, and environment that result from development, deployment, and usage of the software are minimal and/or which has a positive effect on sustainable development [14].

Definition 8: Energy Consumption: The power readings define energy consumption recorded with respect to timestamps. The total energy consumption for the entire run may be simply obtained by summing up the power values measured in *Watts* for the duration of interest. Since each value is the power use of one second, the sum is the energy consumption in Joules [6].

We need to understand the concept of software testing technique that we are using.

Definition 9: Branch Coverage: For achieving branch coverage, each decision should take all possible outcomes at least once either true or false. For example: If(m $\geq$ n), then there are maximum four test cases. These test cases are: 1) m==n, 2) m!=n , 3) m >n , and 4) m<n.

Definition 10: CONCOLIC Testing: Concolic testing is a crossover methodology to programming verification that consolidates Symbolic Execution. It deals with program variables as far as the typical variable for more Concrete Execution, running the program on specific inputs. In this procedure, rest of the system is controlled by introducing standard variables with irregular data and the path condition is achieved alongside regular execution done on the path acquired. New paths are guided by the past paths by flipping or nullifying the last condition seen [4].

III. Proposed Tool Green-JEXJ

To spread the awareness of energy consumption for software testing techniques, we introduced a tool called Green-JEXJ. It may be noted that alone JEXJ module is used to compute branch coverage of Java programs. An power computational tool i.e. JouleMeter has been integrated with JEXJ. So, finally by merging both the modules we obtain Green-JEXJ. The Green by JEXJ concept pursues the objective of reducing the environmental effects in other fields through software engineering and software testing solutions. In safety critical systems such as Nuclear and Aerospace systems, both coverage based testing and energy consumption analysis are necessary. We developed a Java Exclusive-NOR Code Transformer (JEXNCT) to enhance the branch coverage and include it in Green-JEXJ. We have used Java Concolic Unit Test Engine(JCUTE) tool as dynamic symbolic execution tool that has resolved issues such as unavailability of library code and shortcomings of concolic engines. To measure power consumption and energy consumption, we incorporate JouleMeter [6] in Green-JEXJ. We name our framework Green-JEXJ that stands for Green JEXNCT JCUTE.

A. Description of Green-JEXJ

Figure 3 presents the schematic representation of Green-JEXJ tool. Green-JEXJ is the hybrid version of energy consumption and coverage based software testing techniques. Green-JEXJ consists of followings modules: 1) Java Exclusive-NOR Code Transformer (JEXNCT), 2) Java Concolic Unit Testing Engine (JCUTE), 3) Speed Calculator, 4) JouleMeter, and 5) Energy Consumption Calculator.

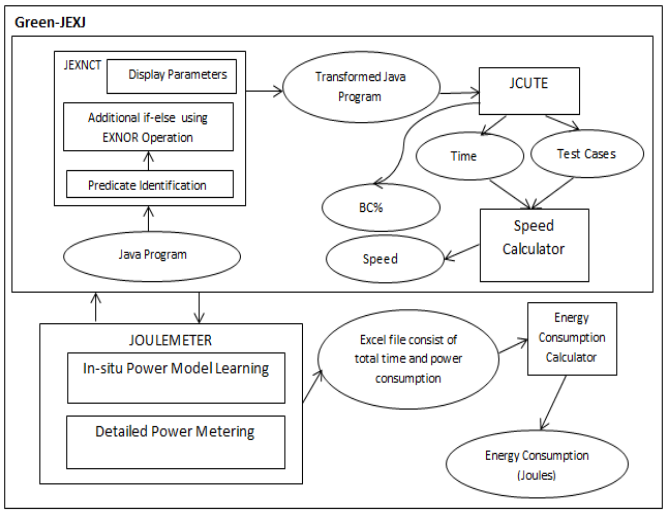

Fig. 3. Schematic representation of Green-JEXJ

The flow starts with JouleMeter. It tracks the testing process and saves the power consumption over the period of time interest in an excel sheet. Initially, JEXJ is executed after supplying a Java program as an input to the JEXNCT. JEXNCT is a code transformer that accepts the target Java program and produces the transformed version of the Java program. JEXNCT consists of two steps: Predicate Identification, and Insertion of empty nested if-else expressions using Exclusive-NOR operation. Putting the original Java program with the generated conditional expressions is termed as the transformed program. Then, this transformed version of Java program is supplied into JCUTE (a concolic tester) to automatically generate test cases and provide branch coverage percentage. We record the total execution time (JEXNCT + JCUTE). Then, Speed Calculator for Test data generation is executed to compute the speed of test case generation. Hence, speed shows the number of test cases generated per minute. After performing this testing process we stop recording JEXJ tool. JouleMeter saved all the values in excel sheet. Since, JouleMeter does not provide energy consumption in Joules, we have developed our energy consumption calculator and plugged into JouleMeter. Energy Consumption Calculator browses the above excel sheet and retrieves timestamps along with total power consumed by the JEXJ tool. Finally, Energy Consumption Calculator produces the total energy consumed during the testing process in Joules.

In Green-JEXJ, two modules i.e. JCUTE and JouleMeter are open source tools and are easily installable. To achieve high Branch Coverage, JEXNCT is plugged into JCUTE. We have developed JEXNCT using Java language to handle Java programs. Since, JCUTE is not capable to measure speed of test case generation, therefore we have developed Speed Calculator that computes the speed of test case generation process. We have used JouleMeter to compute Power Consumption. We have developed a Energy Consumption Calculator and plugged into JouleMeter to produce total energy consumption in Joules as shown in Figure 3.

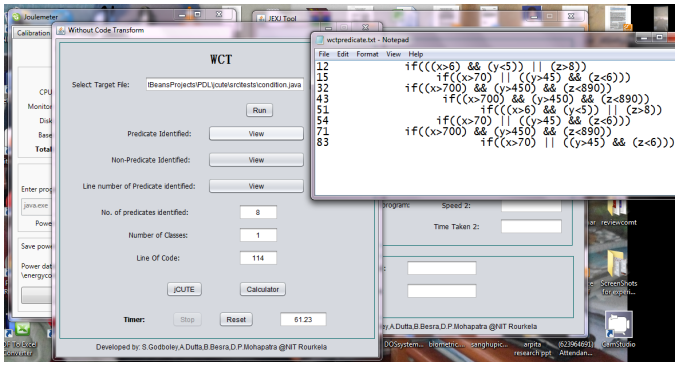

Fig. 4. Number of predicates identified

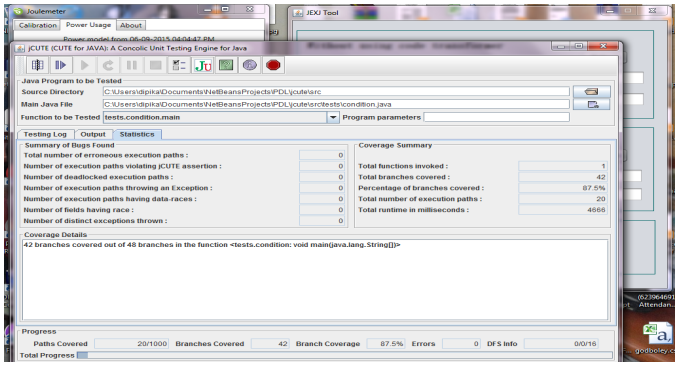

Fig. 5. Execution of jCUTE for original program to measure Branch Coverage

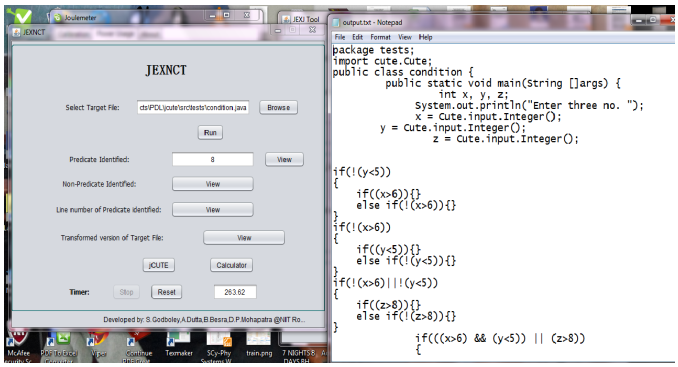

Fig. 6. Transformed version of Java program

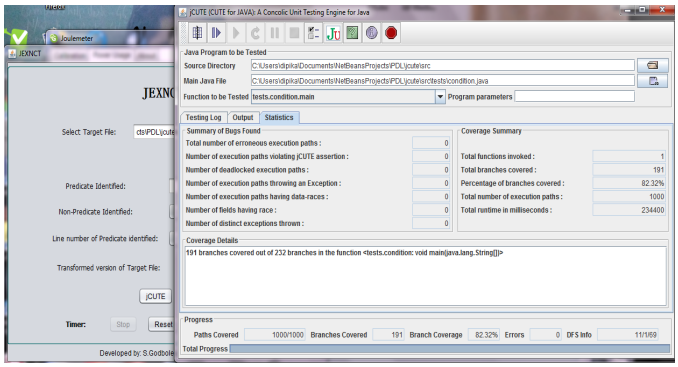

Fig. 7. Execution of jCUTE for transformed program

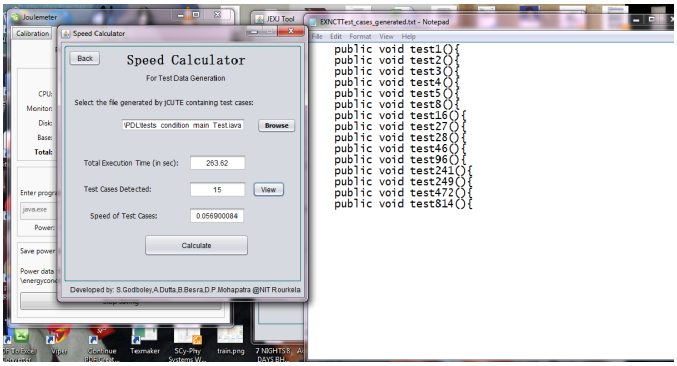

Fig. 8. Execution of Speed Calculator

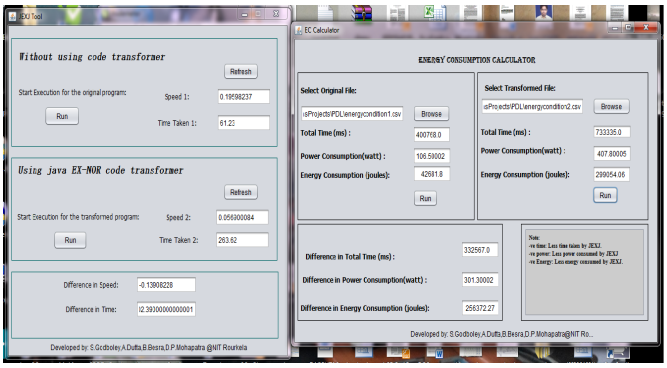

Fig. 9. Final output of JEXJ tool and Energy Consumption calculator

B. Proposed steps for the Green-JEXJ tool

In this section, we discuss the proposed experimental steps to execute Green-JEXJ. Here, we demonstrate the steps through Graphical User Interface (GUI) of Green-JEXJ. These steps are as follows:

Step 1: Execute Power Consumption tool i.e. JouleMeter and set Component Power Usage (Watts). Since, we want to track a software testing tool, so enter the Application Name according to the task manager process tab. Browse a file to save power data and save with a suitable name. Now, start saving the power consumption values by tracking the process that is to be performed.

Step 2: Execute JEXJ tool to perform the testing process. In our testing technique, we need to execute the tool two times. First execution will be without using code transformer (WCT) and second will be with using Java EX-NOR Code Transformer (JEXNCT). Now start execution for the original program, by pressing the 'Run' button.

Step 3: Now to the 'WCT' window will appear. Browse the file to select a target program and click 'Run" button to get parameters such as Predicates Identified, Non-Predicates identified, Line Numbers of predicates identified, Number of predicates identified, Number of classes, and Lines of codes etc. as shown in Figure 4. Press 'view' button to view the identified predicate as shown in Figure 4.

Step 4: To execute JCUTE tool, press the button labeled "jCUTE". Browse the source directory, Main Java file and execute the process to get all the statistics such as Branches Covered, Path covered, Branch Coverage% etc., as shown in Figure 5.

Step 5: Execute the Speed Calculator to measure total speed of test data generation.

Step 6: Stop saving the Power Consumption values for first experiment and browse new file to start saving for second experiment.

Step 7: Now, we perform steps with Java Exclusive-NOR Code Transformer (JEXNCT). Press "Run" button on JEXJ tool for JEXNCT module. Window of JEXNCT will get opened.

Step 8: Browse the transformed program and press 'run' button to get predicates identified, non-predicates, line numbers, and transformed version of the target program as shown in Figure 6. To view the transformed program, press 'view' button as shown in Figure 6.

Step 9: Press "jCUTE" button to execute jCUTE (a concolic tester) to perform concolic testing. Through, jCUTE, we measure the Branch Coverage percentage as shown in Figure 7. In this step, input file will be the transformed one.

Step 10: Execute Speed Calculator for measuring the speed of the test data generation process. Browse test cases file to measure total number of test cases found as shown in Figure 8.

Step 11: Press "Refresh" button to display final values in JEXJ tool. We can observe the difference of time and speed of both the experiments as shown in Figure 9.

Step 12: Stop saving power consumption values for second experiment and execute energy consumption calculator. Browse excel files generated from JouleMeter for both the experiments and hit run buttons to compute and display the final output values as shown in Figure 9.

IV. Experimental Study

In this section, we describe our experimentation on three sample Java programs. We have taken this sample programs from Open System Labrotory [15]. We have already mentioned that our Green-JEXJ tool is the combination of JEXNCT, JCUTE, Speed Calculator, JouleMeter, and Energy Consumption Calculator. JCUTE and JouleMeter are open source tool, and easily installable. We have developed JEXNCT, Speed Calculator, Energy Consumption Calculator and plugged to JCUTE and JouleMeter. All modules are developed in Java language and able to handle Java programs. We have set "Manual Power Model" by providing Base(Idle) power(watts)=65, Processor peak power(high frequency)=35, Processor peak power(low frequency)=10, and Monitor power=50 in JouleMeter.

Since, we are interested to compute power consumption of JEXJ tool, therefore we ignored all the power consumption

TABLE I
CHARACTERISTICS OF JAVA PROGRAMS

Sl. No.	Program Name	LOC	LOC′	Predicates	Function	Classes	BC	BC′	Path	Path′	Error	Error′
1	CAssume.java	64	154	8	1	1	30	98	384	99	3	4
2	Demo.java	56	76	2	1	1	11	28	70	80	77	80
3	weight.java	79	159	6	1	1	28	124	10	63	0	0

TABLE II
OUTPUTS ON EXECUTION OF JEXJ TOOL

Sl. No.	Program Nmae	TC	TC′	Time	Time′	Speed	Speed′	BC%	BC%′	diff_Time	diff_Speed	diff_BC%
1	CAssume.java	13	14	326.98	122.79	2.382	6.84	66.21%	83.33%	-204.19	4.458	17.12%
2	Demo.java	4	6	48.22	63.31	4.974	5.64	87.5%	91.66%	15.09	0.666	4.16%
3	Weight.java	9	14	16.8	54.46	32.142	15.42	87.5%	88.57%	37.66	-16.722	1.07%

TABLE III
OUTPUTS ON EXECUTION OF ENERGY CONSUMPTION CALCULATOR

Sl. No.	Program Name	TT(ms)	TT′(ms)	PC(Watt)	PC′(Watt)	EC(Joules)	EC′(Joules)	diff_TT	diff_PC	diff_EC
1	CAssume.java	614529	271144	442.2	156.5	271744.724	42434.036	-343385	-285.70	-229310.6900
2	Demo.java	114641	141025	17.3	9.5	1983.2894	1339.7375	26384	-7.8	-643.5519
3	weight.java	60924	240398	17.7	13.3	1078.3549	3197.2934	179474	-4.4	2118.9395

of hardware components. We have measured only power consumption of our testing tool through JouleMeter.

Table I shows the different characteristics of the programs considered for experimentation. Columns 3 and 4 shows the size of program in Lines of code. The prime (′) symbol shows the experimental values for the transformed programs. In our experiment, LOCs vary from 56 to 159. Column 5 deals with predicates identified in the programs. Columns 6 and 7 show the Functions invoked and Number of classes present in the programs respectively. Branches Covered, BC and BC′ are shown in Columns 8 and 9 respectively. Columns 10 and 11 deal with total number of paths covered. Columns 12 and 13 show the Errors detected for the original and transformed programs respectively.

Table II shows the outputs on execution of JEXJ tool. Columns 3 and 4 deal with the number of Test Cases(TCs) generated. Please observe that the value of test cases are increased for transformed programs as compared to original programs. Columns 5 and 6 show the time of execution Without Code transformer (WCT) and with Java EXNOR Code Transformer (JEXNCT) respectively. The time is measured in seconds. Speed Calculator is developed to measure speed of test case generation. Columns 7 and 8 show the values of speed of test cases generated, measured in # of TCs generated per minute. Column 9 and 10 present Branch Coverage percentages, BC% and BC′% respectively. Column 11 shows the difference between Time and Time′. Please note that, if difference value is ”-ve”, then it shows that our transformation technique takes less time. Column 12 shows the difference between speed and speed′. Please note that, if difference value is ”-ve”, then it shows that our transformation technique is slower as compared to original one. Column 13 shows the difference between BC% and BC′%. Here, we achieved increase in BC% for all the three programs.

Table III shows the output on execution of JouleMeter and Energy Consumption Calculator. Column 3 shows the total time spent to execute the testing process Without using Code Transformer (WCT) in milliseconds. Column 4 shows the total time spent to execute the testing process using Java EXNOR Code Transformer (JEXNCT), in milliseconds. Columns 5 and 6 show the total Power Consumption of original and transformed Java programs respectively. Power Consumption is measured in *watt*. Columns 7 and 8 deal with total energy consumption for the original and transformed Java programs in Joules respectively. Column 9 shows the difference of total times. Please note that ‘-ve’ value shows that transformed technique takes less time than that of the original technique. Column 10 presents the difference of Power Consumptions for original and transformed techniques. Please note that, ‘-ve’ value shows that the transformed technique consumes less power than original technique. Column 11 shows the difference of total Energy Consumption between the original and transformed techniques. Please note that ‘-ve’ value shows that the transformed technique consumes less energy than that of the original one.

Finally, we achieved 7.45% high Branch Coverage Percentage on an average for the three sample Java programs. We consumed -75945.1008 Joules energy on an average for the three programs on difference of energy consumption.

V. COMPARISON WITH RELATED WORK

In this section we compare our proposed work with some existing related works.

Table IV summarizes the comparison of some related work. We present the framework type developed and used by various authors shown in the third column of Table IV. Brief description of various mentioned research work is provided in the fourth column of Table IV. We can observe from Table

IV that authors listed in sl. no. 1 to 3 proposed their research work based on power consumption and energy consumption. These work help to spread awareness for GREEN IT and GREEN Software Engineering. Authors listed in sl no. 4 and 5 explain about concolic testing and coverage based testing. Last row of Table IV shows our proposed work. We have have presented Green-JEXJ, which is based on Concolic Testing, Branch Coverage, and Green Software Engineering. Green-JEXJ helps to enhance awareness about the importance of energy consumption in software testing.

Table V presents the comparison of different characteristics of the existing approaches. These characteristics are Test Cases, Coverage %, Time Constraints, Speed, Power Consumption, and Energy Consumption, which present that whether these parameters were considered or not in the respective approaches. Among all existing works, only Ding Li et al. [1] have done the analysis of energy consumption for software testing. Please note that authors listed in sl. no. 2 and 3 proposed the work only for energy consumption and power consumption. They have not focused on, software testing technique. Again please note, the authors listed in sl. no. 4 and 5 have only focused on software testing, since they are unaware of Green IT and Green Software Engineering. Last row in Table V shows our proposed work. We have done our research on all the characteristic mentioned in Table V. Our proposed work deals with software testing as well as Green IT, and Green Software Engineering.

TABLE IV
COMPARISON OF DIFFERENT WORKS ON CONCOLIC AND COVERAGE BASED TESTING

S.No	Authors	FrameWork	Description
1	Li et al. [1]	EDTSO	Based on encoding minimization problem as integer linear programming problem
2	Amsel et al. [2]	GreenTracker	Estimates the Energy Consumption of software in order to help concerned uses make informed decision about the software they use
3	Dick et al. [3]	PM,WG,DAE	How to measure the Energy Consumption of software
4	Godboley et al.[13][8]	XNCT,CREST, CA	Approach deals with concolic testing and MC/DC testing. EX-NCT uses Exclusive-NOR operation.
5	Sen et al. [4]	CUTE, JCUTE	Concolic Tool developed for C and Java Programs
6	Proposed Work	Green-JEXJ	Spread the awareness on energy consumption analysis on Software testing techniques

TABLE V
CHARACTERISTICS OF DIFFERENT APPROACHES ON CONCOLIC AND COVERAGE BASED TESTING

S.No	Authors	Generated Test Cases	Measuring Coverage%	Determined Time Constraints	Measured speed	Computed Power Consumption	Computed Energy Consumption
1	Li et al. [1]	✓	✓	✓	X	X	✓
2	Amsel et al. [2]	X	X	X	X	✓	✓
3	Dick et al. [3]	X	X	X	X	✓	✓
4	Godboley et al.[13][8]	✓	✓	X	X	X	X
5	Sen et al.[4]	✓	✓	✓	X	X	X
6	Proposed Work	✓	✓	✓	✓	✓	✓

VI. CONCLUSION AND FUTURE WORK

We proposed a framework named Green-JEXJ to measure the energy consumption of improved concolic testing. We discussed Green-JEXJ along with Fundamental Ideas, the block diagram of Green-JEXJ, and description in detail. We have proposed experimental steps to execute Green-JEXJ. The experimental results show that the proposed approach of test case generation achieved better branch coverage in comparison to the existing methods. Our proposed method achieved 7.45% average increase in branch coverage for the three sample programs. The average difference in energy consumption of the three programs is -75945.1008 Joules('-ve' shows that our transformation approach is energy efficient).

In future, we will extend this work to measure energy consumption of MC/DC testing method. Further, we also aim to develop a distributed framework to increase the scalability of our approach.

REFERENCES

[1] D. Li, C Sahin, J. Clause, and WGJ. Halfond. Energy-directed test suite optimization. *2nd International Workshop on Green and Sustainable Software (GREENS)*, pages 62-69, New York, USA, May 2013.

[2] N. Amsel and B. Tomlinson. Green tracker: a tool for estimating the energy consumption of software. *Proceedings of the 28th International Conference on Human Factors in Computing Systems, CHI*, pages 3337-3342, Atlanta, Georgia, April 10-15 2010.

[3] M. Dick, E. Kern, J. Drangmeister, S. Naumann, and T. Johann. Measurement and Rating of Software Induced Energy Consumption of Desktop PCs and Servers. *EnviroInfo 2011: Innovations in Sharing Environmental Observations and Information*, Shaker Verlag Aachen, 2011.

[4] Sen, Koushik and Agha, Gul. CUTE and jCUTE: Concolic Unit Testing and Explicit Path Model-Checking Tools (Tools Paper). *DTIC Document* 2006.

[5] V. V. Kimbahune, A. V. Deshpande, and P. N. Mahalle. Green Engineering: Future Internet Perspective. *Green Computing CSI Communication: Knowledge Digest for IT Community* Volume(39),Issue(5), pages 10-11. August 2015.

[6] JouleMeter:Computational Energy Measurement and Optimization. http://research.microsoft.com/en-us/projects/joulemeter/.

[7] K. Suryavasnhi, and S. Narkhede. Green ICT in higher Education: The Next Frontier for sustainable growth. *Green Computing CSI Communication: Knowledge Digest for IT Community* Volume(39),Issue(5), pages 12-13. August 2015.

[8] Sangharatna Godboley. Improved Modified Condition/ Decision Coverage using Code Transformation Techniques. *Thesis (MTech) NIT Rourkela* 2013.

[9] S. Kuar, K. S. Dhindsa. Green Computing- Saving the environment with Intelligent use of computing. *Green Computing CSI Communication: Knowledge Digest for IT Community* Volume(39),Issue(5), pages 14-16. August 2015.

[10] Sangharatna Godboley, and Durga Prasad Mohapatra. Time Analysis of Evaluating Coverage Percentage for C Program using Advanced Program Code Transformer *7 th CSI International Conference on Software Engineering* Pages 91-97, Nov 2013.

[11] V. K. Vishwakarma. Green Computing. *Green Computing CSI Communication: Knowledge Digest for IT Community* Volume(39),Issue(5), pages 18-19. August 2015.

[12] S. Godboley, G.S. Prashanth, D.P. Mohapatra, and B. Majhi. Increase in Modified Condition/Decision Coverage using program code transformer. *IEEE 3rd International Advance Computing Conference (IACC)* Pages 1400-1407, Feb 2013.

[13] S. Godboley, G.S. Prashanth, D.P. Mohapatra, and B. Majhi. Enhanced modified condition/decision coverage using exclusive-nor code transformer. *2013 International Multi-Conference on Automation, Computing, Communication, Control and Compressed Sensing (iMac4s)* pages 524-531, March 2013.

[14] S. K. Puri. The concept of Software Recycle. *Green Computing CSI Communication: Knowledge Digest for IT Community* Volume(39),Issue(5), pages 20-21. August 2015.

[15] Open System Labrotory. URL: http://osl.cs.illinois.edu/software/jcute/.