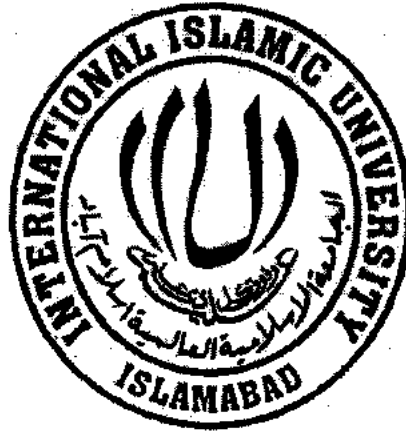


GUI Test Path Coverage and Optimization Using Ant Colony Optimization



Submitted By:

Mashal Ibrar

291-FBAS/MSSE/F09

Supervisor: Dr. Aamer Nadeem

Associate Professor DCS, MAJU

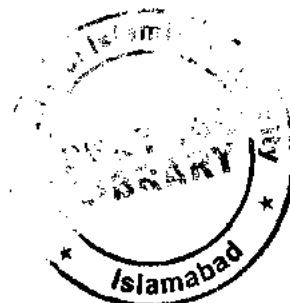
Co-Supervisor: Muhammad Imran Saeed

Department of Computer Science and Software Engineering

Faculty of Basic and Applied Sciences

International Islamic University Islamabad

2014



Accession No. JH12912

K

dp

Size
- 300dpi
- ocr
- smulpdf.

DATA ENTERED

MS

003

MAGS

optimization (Systems)

system design

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

Final Approval

INTERNATIONAL ISLAMIC UNIVERSITY ISLAMABAD DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

Date: 28-08-2014

Final Approval

This is to certify that we have read the thesis submitted by Miss Mashal Ibrar, Registration # 291-MSSE/FBAS/F09. It is our judgment that this thesis is of sufficient standard to warrant its acceptance by International Islamic University, Islamabad for the degree of MS in Software Engineering.

COMMITTEE

EXTERNAL EXAMINER

Dr. M. Sikandar Hayat Khiyal,
Professor,
Faculty of Computer Sciences
Preston University, Islamabad



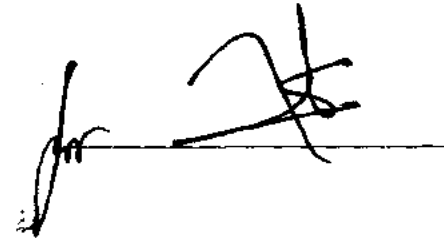
INTERNAL EXAMINER

Mrs. Zareen Sharf,
Assistant Professor,
Department of Computer Science & software Engineering,
IIU, Islamabad



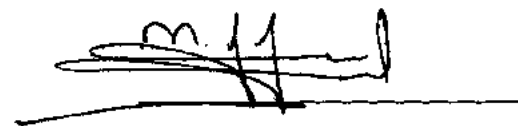
SUPERVISOR

Dr. Aamer Nadeem,
Associate Professor,
Department of Computer Science,
Mohammad Ali Jinnah University, Islamabad



CO-SUPERVISOR

Mr. Muhammad Imran Saeed,
Assistant Professor,
Department of Computer Science & Software Engineering,
FBAS, IIU, Islamabad



**A dissertation submitted to the
Department of Computer Science and Software Engineering,
Faculty of Basic and Applied Sciences,
International Islamic University, Islamabad as a partial
fulfillment of the requirements for the award of the MS (SE)
degree**

Dedicated To

My Family

DECLARATION

I, Ms. Mashal Ibrar hereby declare that this thesis, neither as a whole nor as a part has been copied out from any source. It is further declared that this dissertation is entirely based of my personal efforts made under the sincere guidance and help of my supervisor. No portion of the work presented in this report has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

Mashal Ibrar
Registration#291-FBAS/MSSE/F09

ACKNOWLEDGEMENT

First of all I would like to start with the name of ALLAH who has given us the ability and understanding that we came this far and we were able to complete my thesis. I pray that ALLAH keep us guiding like this through every stage of our life.

I am extremely thankful to my supervisor, Dr. Aamer Nadeem, who was the real source of encouragement and motivation during the whole work of thesis. His useful suggestions, advice and ideas to bottleneck problems encountered during this thesis work were just immeasurable.

My appreciation's also goes to Muhammad Imran Saeed for his precious guidance, fruitful discussions and encouragement throughout this thesis work. I am thankful for his kindness, patience and feedbacks. His expertise, devotion and constant encouragement was very helpful and made this effort an enjoyable one.

At last, I am thankful to my family especially my Mother and Sister for her kind support and encouragement. Indeed, without their prayers, emotional and financial support it would not be possible for me to accomplish this work.

Mashal Ibrar

Registration # 291-FBAS/MSSE/109

ABSTRACT

Graphical user interface (GUI) has become imperative and commonly used to interact with software system. GUI is a type of user interface that interact with system by the user. GUI contains approx. 50-60% [6] [34] of the total software code. However GUI make ease to the user to interact with the software but the development process is becoming complex day by day due to large number of GUI interactions. A software organization aims to produce a high quality software product which can be accomplished through testing from various perspectives.

GUI testing is a critical activity that is designed to find defects in the GUI of overall application, and aims to produce reliable, accurate and cost effective system .GUI testing is a system level testing in which event sequences are tested to validate that the desired functionality is full filled or not. Sequences of events are tested against the functionality. Model based software testing (MBST) are used to generate test paths automatically by traversing the model. However test paths generated from these models may be feasible or infeasible.

In MBST, Event Flow Graph (EFG) is used to generate test paths efficiently. In EFG model GUI objects are denoted by events and the edges between events shows the dependencies between events. The testing needs to be done in such a way that it provides effectiveness, efficiency, improved fault detection rate and maximum coverage. To cover all test paths and events of EFG, a technique is proposed that is used to test and generate all paths of event flow graph using ant colony optimization optimally. Another challenging question in the software testing is that how much testing is enough to achieve maximum fault tolerant software product. A better coverage criterion is use to answer that how much testing is required. Simple graph traversal algorithm generates both feasible and infeasible test paths. Infeasible test paths leads to unknown states which cannot execute. Ant Colony Optimization (ACO) algorithm is also used along with the Event Flow Graph to generate feasible test paths optimally. Due to large and complex nature of software system, testing is done in minimum time to achieve full coverage through event flow graph using ant colony optimization algorithm. The coverage criterion is used to measure software quality by testing the whole system. This ensures that maximum coverage is achieved from event-interaction coverage criterion as compared to simple event coverage criterion. The proposed approach generate feasible test paths of all events and all event-pair interaction.

Table of Contents

1	Introduction	10
1.1	Software Testing:	10
1.1.1	Test Paths	11
1.1.2	Test Data Generation	11
1.2	Graphical User Interface Testing:	11
1.3	Event Flow Graph:	12
1.4	Research objective	12
1.5	Organization of Thesis	13
2	Background	15
2.1	GUI TESTING:	15
2.2	Code coverage:	16
2.3	Graphical user interface (gui) coverage	16
2.4	Event Flow Graph:	17
2.5	Types of Gui events	18
2.6	Types of Event Coverage:	19
2.7	Optimized Test data generation:	21
2.7.1	ANT COLONY OPTIMIZATION:	21
3	Literature Survey	26
3.1	State-Based Techniques:	27
3.2	Event Flow graph	29
3.3	Genetic algorithm	30
3.4	Ant colony optimization	32
3.5	Coverage Criteria	34
3.6	Comparison of existing techniques	35
4	Problem Statement	38
4.1	Limitation on existing methodologies:	39
5	Proposed Approach	41
5.1	Research Approach	42
5.2	Ant colony optimization Technique	43

5.2.1	Parameter setting.....	44
5.3	Proposed Approach.....	45
5.4	proposed approach algorithm.....	48
5.5	Methodology used to prove	49
6	Implementation	51
6.1	Tool architecture.....	51
6.2	Description about tool architecture	51
6.3	Graph representation.....	52
6.4	implementation details.....	56
7	Case Studies.....	62
7.1	Description about example.....	62
7.2	generated test paths	64
7.3	Case study 2	70
7.4	Experimental Results of event coverage and event-interaction coverage	70
7.5	Case study 3	74
7.5.1	Generated Test paths	75
7.6	Case study 4	76
7.6.1	Generated Test paths	77
8	Results and Discussion.....	82
8.1	Analysis of Experimental Results	82
9	Conclusion and future work.....	88
9.1	Conclusion.....	88
9.2	Future Work.....	88
10	References	91

List of Figures

- Fig 2.1 Example of Graphical User Interface (GUI)
- Fig 2.2 An Event Flow Graph (EFG) for a part of MS WORD
- Fig 2.3 Example of Restricted Focus Event
- Fig 2.4 Example of Unrestricted Focus Event
- Fig 2.5 EFG before event coverage
- Fig 2.6 EFG after event coverage
- Fig 2.7 EFG before event-interaction coverage
- Fig 2.8 EFG after event -interaction coverage
- Fig 2.9 Pheromone trail of real Ants approach
- Fig 2.10 Ant Colony Optimization Algorithm Flowchart
- Fig 3.1 Example of simple finite state machine
- Fig 5.1 Abstract level Research Approach
- Fig 5.2 Diagram of Proposed Approach
- Fig 6.1 Tool Architecture
- Fig 6.2 GUI of proposed framework
- Fig 7.1 Event Flow Graph (EFG) Example of Notepad
- Fig 7.2 Comparison of event coverage in existing and proposed approach
- Fig 7.3 Comparison of edge coverage in existing and proposed approach
- Fig 7.4 EFG of Internet Explorer (IE)
- Fig 7.5 Coverage criterion of proposed approach
- Fig 7.6 EFG of windows media player
- Fig 7.7 EFG of windows Gtalk
- Fig 8.1 Comparison of Event Coverage and Event-interaction Coverage
- Fig 8.3 Coverage criterion of proposed approach for Case study 2

List of Table

Table 3.1 Comparison of different approaches

Table 5.1 ACO Parameter Setting

Table 7.1 Comparison of event coverage

Table 7.2 Comparison of Event-interaction Coverage

Table 7.3 Event Coverage of Internet Explorer (IE)

Table 7.4 Event-interaction Coverage of Internet Explorer (IE)

Table 8.1 Comparison of Experimental Results

Table 8.2 Coverage criterion for internet explorer

Chapter 1

Introduction

1 INTRODUCTION

Software Testing is the major and important part of software development. Testing takes almost 50-60% [6] [34] of effort and cost of the software development. A test case normally encompasses of an input, output, anticipated result and the actual result. A collection of test cases are called test suite. A test suite contains goals and objectives of each test case. More than one test case is required to test the whole functionality of the GUI application. Due to the importance of the testing phase in a software developmental lifecycle, testing has been divided into graphical user interface (GUI) based testing, logical testing, integration testing, unit testing and so on. GUI Testing has become very important as it provides ease of use to user to interact with the software. As the time passes the complexity of GUI testing is increased. The basic aim for the software testing is to provide effectiveness, correctness, better fault detection rate and maximized coverage. Simple testing techniques are used to generate test cases but the generated test cases are might be feasible and infeasible. Due to complex and real nature of software, numbers of generated test cases are infinite. Ant colony optimization is used to generate the optimized and feasible test cases and for finding shortest path which has been purposed to overcome the limitations of finite test cases.

1.1 SOFTWARE TESTING:

Software testing is done through various methods like manual testing and automated testing techniques. In Manual testing, tools such as capture and replay, scripts-driven, and data-driven approach test cases are generated but there are numerous defects exists due to difficulty of massive manual work, low adaptability to software variation, and lack of management for test cases and their coverage. Manual GUI testing is done by tester manually and having a more chance of error in it. Manual testing is tremendously slow, time consuming and expensive. After that automated technique was introduced that enables the process of generating test cases automatically. Automated GUI Testing consist of automated testing tasks that have been done manually before, using automated techniques and tools. It is more effective, reliable, accurate and cost effective [5] [9] [28].

1.1.1 Test Paths

Test path is the execution of event sequence from source to the target point. The output of a program may have a large or infinite number of paths. Input is given to the program, as a result its expected outcome and generated outcome have to examine. Feasible and infeasible are two types of Generated Paths [17]

- **Feasible Paths:** A path for which a collection of input values are given resulting the program to be executed.
- **Infeasible Paths:** A path that cannot be executed by some set of possible input values

1.1.2 Test Data Generation

In automated software testing sufficient test data generation is the method of finding a set of program input data, which satisfies a given testing criterion [4]. Automated testing is used to generate quality wise product in a low cost. For achieving this functionality test data generation techniques such as random, symbolic, and dynamic test data generation techniques exist. Swarm optimization techniques have been applied recently to generate test data successfully.

1.2 GRAPHICAL USER INTERFACE TESTING:

Graphical user interface testing is the testing process of software's graphical user interface to preserve and achieve fault free software. In GUI testing, some tasks are performed and then the actual result is compared with the predictable output. GUI testing is a critical activity that is designed to find faults in the GUI or in the whole application, and increasing the self-assurance in correctness which gives reliable, accurate and cost effective system .GUI testing is a system level testing in which event sequences are tested to validate that the desired functionality is full filled or not. Sequences of events are tested against the functionality. Model based GUI testing (MBST) are used to generate test paths automatically by traversing the model. However test paths generated from these models may be feasible or infeasible.

Li et al. [15] discussed in their research work that there are some important characteristics of GUI systems and their limitation are described: [15]

- In GUI there are extremely large number of input and events which indicates to large number of states to be tested. Large number of states leads testing complex and challenging.
- The synchronizations and dependencies between objects cannot be controlled in the similar window (e.g. object in one window may be linked to an object in another window).
- Object oriented software programming
- In GUI applications the user may use a keyboard shortcuts, a button click, a menu option, a click on another window etc. How many of these should be tested.
- If the windows are closed before finalizing a transaction may leave the application or the database in an inconsistent state.

1.3 EVENT FLOW GRAPH:

To model GUI component and objects, Event-Flow Graph (EFG) is commonly used. Event-Flow Graph represents all user interactions between the events in a GUI component. In Event-Flow Graph, events are represented as vertices (nodes) and the relationships or interaction between events are represented as edges (arrows) connecting pairs of event vertices. The Event Flow Graph used for Automated GUI testing.

1.4 RESEARCH OBJECTIVE

Our main objective is to generate test path which cover all the events and all event-interaction coverage in event flow graph. We will improve efficiency of test data generation. We will generate the test paths in less number of test execution and less time by calculating the probability. Existing problem is that the generated test paths didn't cover all events and their child relations (follow-relation). So, we enhance the coverage of all events. In this research work an automated test path generation framework is proposed for GUI testing. The main objective is to generate all possible events and event-

interaction relationship dynamically by using Ant colony algorithm. Also we eliminated the infeasible test path generation whereas the test path generation is automatic. After the experimental results it has been proved that proposed solution covers all events and all event-interaction (follow-relation events). The generated test path from the proposed solution also remove the infeasible test path.

1.5 ORGANIZATION OF THESIS

This thesis division is as follows. Chapter 2 gives the detail background knowledge of automated software testing and Ant Colony Optimization algorithm. Chapter 3 includes the related works of automated test data generation techniques and test data generation from event flow graph using ACO. This chapter includes the comparison of existing approaches. Chapter 4 include the problem definition which was deduced from literature survey. Chapter 5 provides the proposed solution of the problem which is discussed in literature survey. Chapter 6 include the Implementation details and tool architecture of the proposed solution. Chapter 7 contain the case studies of different GUI examples on which proposed approach is applied. Chapter 8 is comprised of result and discussion which explains and evaluates the experimental results of the proposed approach. Chapter 9 concludes the thesis work and gives the future work.

Chapter 2

Background

2 BACKGROUND

This chapter includes background of automated GUI testing techniques in detail, coverage criteria and details of Ant colony optimization that are used to achieve accurate and optimize test data generation.

2.1 GUI TESTING:

GUI testing is vital to make the entire system safer and more reliable. Example of GUI is shown in Fig.2.1 which shows how the user interact with the software. GUI testing is a challenging activity aimed at finding defects in the GUI application to increase the confidence on its correctness and accuracy. Software testing techniques can be categorized into two main types i.e., static testing and dynamic testing. In Static testing, program is not executed which is used to find errors by reading the code and examination the design as non-execution or verification technique performed by tester or automated tool. This technique cannot perform detailed testing. Some static analysis methods are code inspections, code walkthroughs, and code reviews. Static testing is more error prone, extremely slow and unacceptably expensive [5].

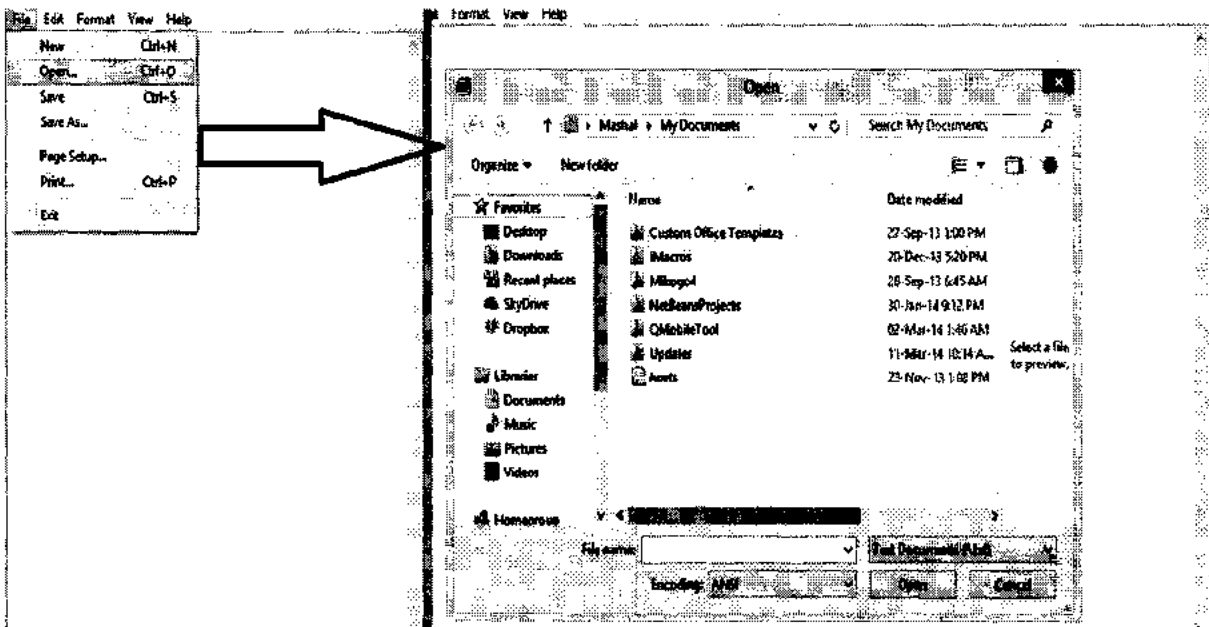


Fig 2.1 Example of Graphical User Interface (GUI)

Dynamic testing is known as validation technique. This technique must execute the code and verify the output with expected outcome. Dynamic testing has further classified into two parts, i.e., black box testing and white box testing.

2.2 CODE COVERAGE:

Code coverage is a technique to ensure that code must be tested through test cases. For accurate software, coverage criteria are used to achieve fault-free software. Coverage criteria are sets of rules that are used to define coverage of GUI components for adequate program. The most renowned code coverage criteria are statement coverage, branch coverage, decision coverage, decision condition coverage and path coverage.

In Statement Coverage criteria every statement in the program need be executed at least once. Statement coverage is known as a weaker criterion than others because it doesn't ensure that it executes the same statement in different sequence [29].

2.3 GRAPHICAL USER INTERFACE (GUI) COVERAGE

For GUI testing various models are used to represent the GUI components as events or nodes and relation between them is shown in dependency. For GUI testing some models are used for testing know as Model based testing (MBST) .In MBST models are traversed to generate test paths. For Model Based GUI testing different coverage criterion are proposed. One of them is path coverage criterion. Path Coverage Criterion must execute all the paths from the starting to the end node in the flow graph.

Due to complex nature of software there are various paths in a program having loops resulting infinite test paths which didn't ensure the accuracy of a tested program through path coverage. This means testing in a finite period of time is very challenging due to time pressure and scarcity of other resources, software testing must be fulfilled within fixed time period [2].

A path coverage criterion is stronger than branch and statement coverage because it covers all sub sequences of a program.

2.4 EVENT FLOW GRAPH:

To model GUI component, Event-Flow Graph (EFG) is commonly used. Event-Flow Graph symbolizes all interactions between the events in a GUI component. Memon et al. worked a lot on the GUI testing along with event flow graph. In Event-Flow Graph, events are represented as vertices (nodes) and the relationships between events are embodied as edges (arrows) connecting pairs of event vertices as shown in Fig 2.2. The Event Flow Graph used for Automated GUI testing.

An Event-Flow Graph contains all events that may be executed at a given time. When GUI components are dynamic, it may be accessed at the same time when parent of GUI component are accessed. Event-Flow Graph always contains events for a GUI component along with all of its child (adjacent) GUI components. In a typical GUI component, there is a high connectivity between GUI events.

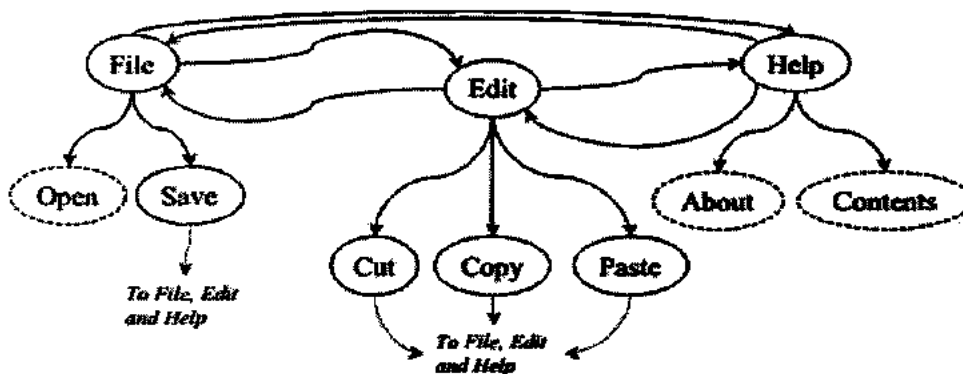


Fig. 2.2 An Event Flow Graph (EFG) for a part of MS WORD [8]

However above coverage criterion do not ensure the adequacy of GUI test cases for following reasons. Firstly, the source code of pre-compiled elements may not always be available to be used for coverage criterion. Secondly, GUI input consists of sequence of user events and the number of possible variations of the events that may lead to a large

number of GUI states. For adequate testing, GUI events may be tested in a large number of states. Due to high level of abstraction GUI event sequence can't be obtained from code. Similarly the code can't be used to guarantee the adequacy of the sequence of events that have been tested.

Memon presented some contribution related to the characteristics of the coverage criteria [6] to overcome the above challenges of the coverage criteria based on events in GUI. GUI events are divided in further sub groups.

2.5 TYPES OF GUI EVENTS

- **Restricted-focus events (Modal windows)** the windows that once opened, they control the GUI interaction, restrict the user to a specific range of events within that window until the window is terminated by a termination event explicitly.

Open menu of file menu in MS Word is an example of restricted-focus events in GUI systems where the user clicks on open button, a new window appear and the user select and customize the options, and explicitly terminates the window by either clicking Cancel or Open as shown in Fig 2.3

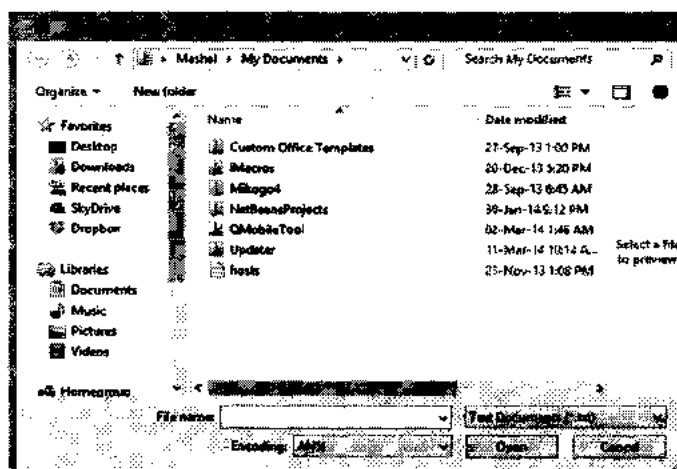


Fig 2.3 Example of Restricted Focus Event

- **Unrestricted-focus events (modeless windows)** the window that do not restrict the user's within that window are unrestricted events. Note that the difference

between restricted events and unrestricted-focus events is that the restricted windows have to be explicitly ended.

For example in the MS Word, open Edit menu then select Find are displayed in an unrestricted-focus window as shown in Fig 2.4

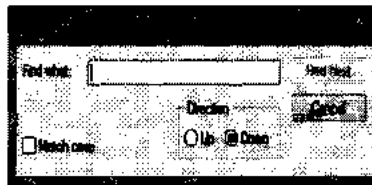
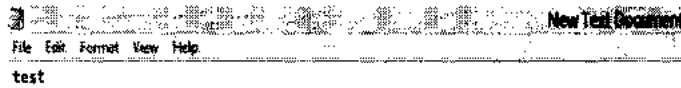


Fig 2.4 Example of Unrestricted Focus Event

- **System-interaction events** interact with the software to accomplish some operation. Examples are cutting and pasting text, and opening object windows.
- **Termination events** terminates modal windows. Examples are Ok and Cancel.

2.6 TYPES OF EVENT COVERAGE:

The are two main types of coverage criteria for events which are as follows

1. **Intra-component Coverage:** Intra coverage criteria contain Event Coverage, Event-interaction Coverage (event pair coverage) and Length-n Event-sequence Coverage. Detail of these coverage criteria is described below.
 2. **Inter-component Criteria:** Inter coverage criteria divides into following category: Invocation Coverage, Invocation-termination Coverage and Inter-component Length-n Event-sequence Coverage. In this paper we only consider Intra-component coverage criteria.
- **Event Coverage**

In Event coverage all event in the component must be executed at least once. It is essential to check that all event are executed as expected. When all the events are executed at least once, event coverage criteria is achieved.

For example In Fig 2.5 Empty circle shows events sequences which is not executed. In Fig 2.6 filled circle shows that events are executed or traversed at least once.

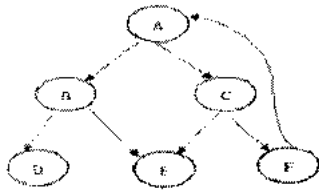


Fig 2.5 EFG before event coverage

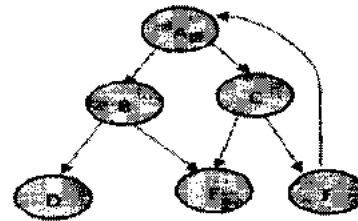


Fig 2.6 EFG after event coverage

- Event-interaction Coverage

Event-interaction criteria is also known as event-pair coverage criteria. In GUI testing it is essential to verify the interactions among all pairs of events in the module. For achieving event interaction coverage the pair of events may be executed in a sequence.

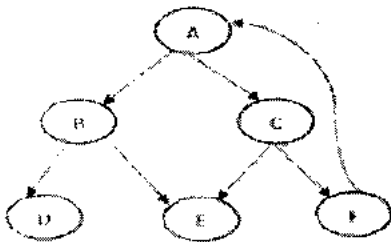


Fig 2.7 EFG before event-interaction coverage

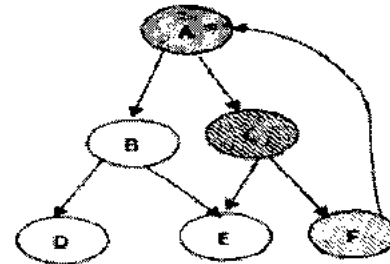


Fig 2.8 EFG after event -interaction coverage

In this criterion all event- interaction of event A should be executed at least once after an event A has been performed, like F are executed at least once than event-interaction criteria for event A is fulfilled as shown in Fig 2.8. Event is set as completely explored when all its incident events are executed at least once. Lines in events shows that events

are executed at least once. When all its incident node are executed its color changed into solid blue like in event A.

- **Length-n Event-sequence Coverage**

Sometime different contexts may change the behavior of events .In that situation event coverage and event-pair coverage criteria is not appropriate for sufficient testing. For this purpose a criterion is defined that captures the contextual impact formally. As the length of the event-sequence increases, the number of contexts also increases.

2.7 OPTIMIZED TEST DATA GENERATION:

State-based testing is commonly used in automated GUI testing. Test data generation is very crucial in software testing. Test suite generation does not detect the errors in software but also ensure cost reduction associated with software testing. State-based testing is a normally used approach in GUI testing. There are two main problems related to state-based software testing:

- (1) Some infeasible test case are generated.
- (2) Some redundant test data are generated to achieve the necessary testing coverage.

2.7.1 ANT COLONY OPTIMIZATION:

Software testing is one of the major part in the software developmental life cycle. Due to cost, time and other environment, exhaustive testing is not feasible and selecting the right test path is one of the problem in software testing. To overcome these problem we need to automate testing process and generation of effective test paths can decrease the overall cost of testing as well as chance of finding the defects in the software system. For this situation we need to apply ant colony in our real software system to generated feasible and optimize test paths in less time.

Ant Colony Optimization (ACO) is a meta-heuristic approach motivated from the behavior of real ants. The approach seeks ants to discover the shortest path to the food source

with the help of a chemical substance called pheromone. Due to time and other resource scarcity it is the need of the software to be adequately tested in an optimized way. For this purpose ACO technique is used to obtain optimized test data generation which covers all events and event-pair interaction.

Working of Ant Colony:

According to [32] the working of ACO is as follows

- The ants walk from the nest to the food source while leaving a substance called pheromone on their path.
- Pheromone acts as a guidance to choose their paths depending upon the stronger pheromone value.
- Pheromone trail is made where the pheromone is deposited. This trail allows other ants to find the sources of food that have previously acknowledged by other ants showing in Figure 2.9.

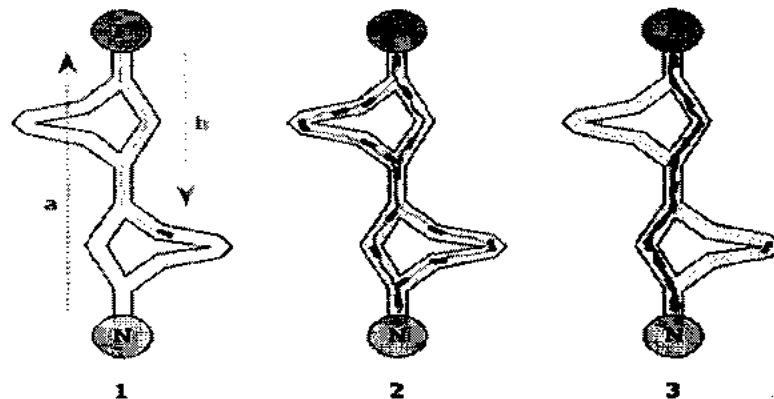


Fig 2.9 Pheromone trail of real Ants approach [35]

- With the passage of time pheromone continuously evaporate at some exact rate. The ant which cover shorter path would return first to the nest. Those paths which have high probability the choose of that path is high. At the end all the colony ants converge to follow the shortest path after some time which is shown in figure 2.9 last path where all the ants are moving through the shortest path.

To construct probabilistic solutions, the pheromone trails reflects developed search experience of ants and heuristic information related to the problem.

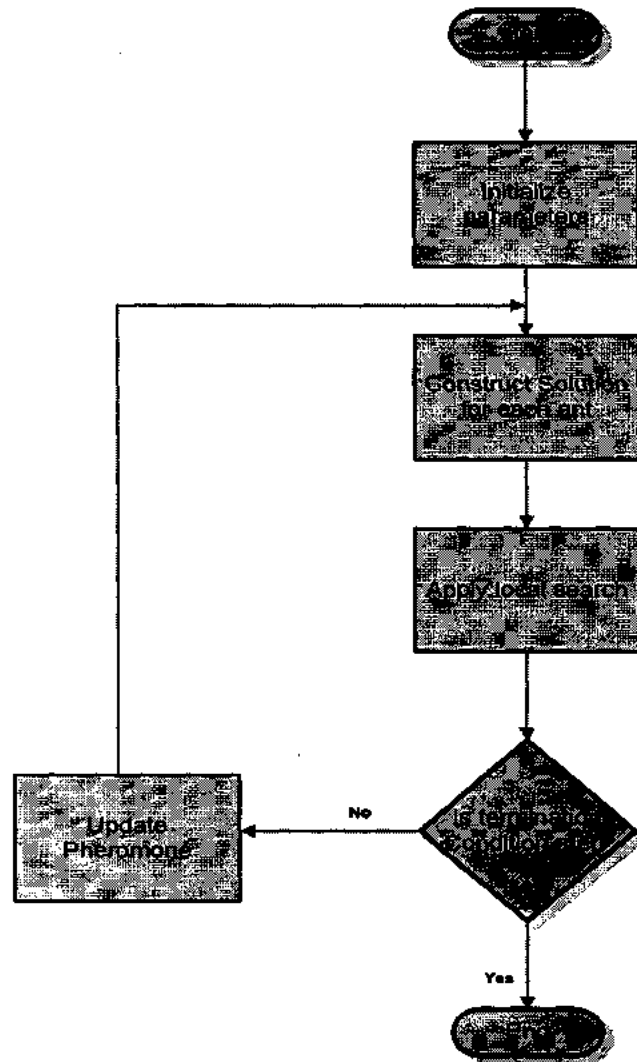


Fig 2.10 Ant Colony Optimization Algorithm Flowchart

In Ant colony optimization, Local search algorithms is means to find the best and optimal solution and find it till not found. It start from a complete initial solution and try to find a better solution in suitably defined neighborhood of the current solution. As represented in

Fig 2.10 for solution improvement, the algorithm searches the neighborhood. If improved and optimal solution is found, it replaces the current solution and the local search continues until no improving neighbor solution is left behind in the neighborhood of the current solution and the algorithm ends in a local optimum.

Chapter 3

Literature Survey

3 LITERATURE SURVEY

This chapter includes the literature survey of automated test data generation using finite state machine (FSM) and event flow graph (EFG). As mentioned the research problem in above chapter 1, the generated paths are grouped into feasible and infeasible test paths. Some of automated test data generation techniques produces infeasible data.

In the early age of test data generation author used manual testing techniques and strategies. These techniques requires more time and effort and the produced system are more error-prone. To overcome the limitation of manual testing some authors used the specified models to represent the GUI interface. Then test data is generated from these models. This acquire less effort and cost related to testing.

Many researchers have been worked on automated test data generation as in [25] proposed the approach of test data generation based on finite state machine. Major research have done on State diagram for automated test data generation. Several variation of state machine have been used for software testing, such as Finite State Machine Models (FSM), UML Diagram-based Models and Markov Chains.

Due to large and complex software system problem of large input-space have faced. To resolve this situation Event flow graph (EFG) are used. GUI software have different interaction with user and other events of GUI. To represent these interactions between events, EFG are used for GUI testing.

Many researcher have worked on the test data generation optimally. To gain this functionality many swarm optimization techniques are used. Genetic algorithm (GA) and Ant colony optimization (ACO) are related to swarm optimization to test software system optimally.

Here are some techniques that is used to generate test data automatically proposed by many researchers.

3.1 State-Based Techniques:

Many researchers have been worked on automated test data generation. The most commonly proposed approach is test case generation through finite state machine and its variation. To construct a state machine model, assume GUI behavior as a state machine. State transition in the state machine shows the input events.

3.1.1 Belli et al. [7]

Finite state machines have been used to model GUIs .Inside GUI there are different user interactions and interactions between states. GUI states are represented as windows and interaction is consider as a transition in the FSM. Test case or test path in GUI testing corresponds to the sequence of user events. In this paper the author converted FSM into simple formal expressions. The formal expression were used to generate event sequences.

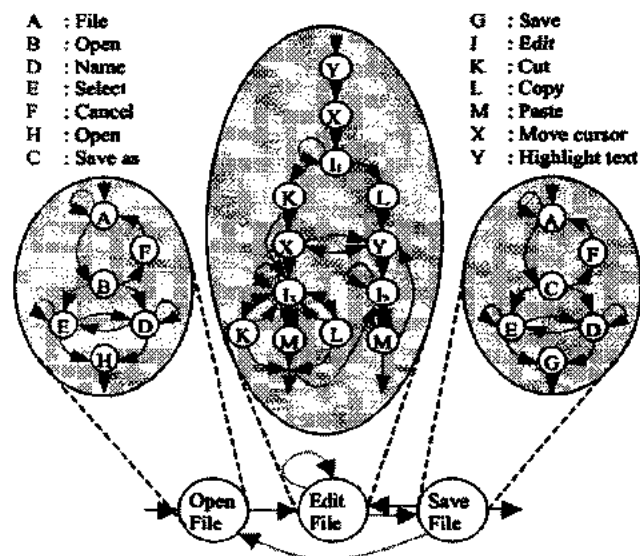


Fig 3.1 Example of simple finite state machine [7]

Limitation :

Due to the large number of possible states as shown in figure and Complex GUI events, FSM have faced scaling problems.

3.1.2 Memon and Sofa et al. [8]

Memon et al. used Artificial intelligence (AI) to accomplish the state-space problem by eliminating unambiguous states. Software tester constructs description of GUI states manually. Planning operators gives the description that defines the preconditions and post-conditions of every GUI event. Test cases are generated automatically by starting a planner which traverse the path from start to the target state. This methodology work fine for small command language system.

Limitation:

Complex GUI have a large number of actions like windows, buttons and menus. So this approach needs to be enhanced to manage large number of operators.

3.1.3 White et al. [10]

White et al. divides state space problem into complete interaction sequences (CIS) by using state machine. The test designer categorizes each user action into CIS. The CIS is used to generate test cases. This approach is effective for unit testing that divides the whole GUI into small functional units.

Limitation:

Huge manual effort is required in modeling the finite state machine for testing, especially when the program is not implemented.

Modeling of state model require executive resources.

The state machine depend upon the understanding the models according to test designer. When GUI applications are larger and complex it is difficult to manage and analyze.

Integration testing is not done through this methodology.

3.1.4 Jin and Wang et al. [22]

In this paper Finite State Machine works with operational profile (op). Probability of random selection of input creates the OP. Then generate the test paths on the basis of these operational profiles. At Last the process of validating the effectiveness of this method is measured through design experiment.

Limitation:

The combinations of GUI objects state spaces are enormous, and it is impossible to test all of them. If the object is in mediate complicated, it is almost impossible to take the advantage of finite state model. There might be chance of choosing incorrect operational profile. Only unit testing is done in this case.

TH
129

3.2 EVENT FLOW GRAPH

To overcome the state-space problem in finite state machine and generation of large number of states in large and complex software system, many authors generated the test data using Event- Flow graph.

3.2.1 Memon [16]

Many authors used different graph models to generate detailed test cases. Memon associate all of the models into event-flow model and generate test cases from theses models from implementation [16]. GUI ripper is used to automatically generate EFG that represents all possible event sequences of GUI. The model overcome the event-space exploration problem for GUI testing.

Limitation:

Size of the space of all possible event interactions grows exponentially with length.

3.2.2 Lu and Wang et al. [19]

Lu and Wang et al. proposed GUI automation test model based on the event flow graph (EFG). In EFG model, a methodology is proposed to generate test cases using smoke testing using an improved ACO.

On the other hand, spanning tree obtained by deep breath-first search (BFS) approach is used to generate test cases from initial point to target point.

Limitation:

GUI automation test software needs to verify the validation of the model automatically.

Lack of adaptability of the various GUI Operating system.

Event-flow graph needs improvement to explain the complex logic problem and reduce the involvement of manual verification.

3.2.3 Memon and Yuan [23]

Memon and Yuan presented automated model based technique used to generate test cases by using feedback techniques. The test cases in the seed test suite are aimed to generate test cases automatically and efficiently. GUI runtime information is used as feedback to generate test case iteratively. This technique can heal infeasible test cases used by feedback information.

Limitation:

This approach is complicated and expensive to generate models from Event Flow Graph to Event Interaction Graph and then from Event Interaction Graph to event semantic interaction graph.

3.3 GENETIC ALGORITHM

Genetic algorithm are also used for GUI test data generation. Some of the contribution of the researcher are following.

3.3.1 Rauf et al. [24]

Genetic algorithm finds the best possible combinations that are related to some test criteria. This criteria measure how much coverage is achieved by coverage function. This methodology include following steps.

Start: It generate a random population of n chromosomes.

Fitness: Evaluate the fitness of each chromosome x in the population.

New population: Create a new population by selection, crossover and mutation then acceptance of the new populated off spring.

Fitness Function: In fitness function input is given then the fitness function produce the result which presents the acceptability of the program.

Rauf et al. [24] used Genetic Algorithm to seek optimal test data for GUI testing. Genetic Algorithm has been used for the test coverage optimization. A genetic algorithm is suitable for nonlinear complex problems.

Limitation:

Manual test data generation through clicking on various GUI elements.

Increase in the number of generations for improving coverage which is time consuming and slow.

3.3.2 Preeti et al. [27]

UML state chart diagram using Genetic algorithm are used to generate optimal test cases. To generate new test sequence, crossover has been applied and productivity of the test sequences is calculated by Mutation Analysis. Generate the State flow diagram and collect all the possible paths between the starting to the ending State. Select two possible paths and then perform crossover on these selected paths. It will generate a new path after that mutation is applied on this new path to eliminate the dead paths or infeasible paths.

Limitation:

If the number of test sequences are less the result should not be ideal. Only suitable for complex and real time application

Expert's user's assumption uses longer test paths through various input event interaction when performing different functions or actions as compared to the novice user

3.3.3 Khamis et al. [34]

This paper presents a new general technique for the automatic test data generation for spanning sets coverage. The proposed methodology generates program units by spanning sets to gain test coverage criteria automatically. Spanning sets are covered to generate test data automatically. The GA starts by creating an initial population of individuals randomly. Crossover and mutation is done on these population set to obtain the required test data. The fitness function is calculated in some problem-dependent way.

Limitation:

The problems of infeasible paths identification and train the system to avoid the infeasible paths during finding the optimal solution.

3.4 ANT COLONY OPTIMIZATION

In the field of GUI testing Ant colony optimization is also used to generate test path optimally.

3.4.1 Li and P. Lam [12]

This paper proposed an Ant Colony Optimization approach for state based test data generation. The advantages of the proposed approach are feasible, non-redundant data generation. ACO depends on probabilistic technique that can be useful to generate combinatorial optimization solution. To represent the State chart model a directed graph is created. By using Ant Colony Optimization algorithm, a cluster of ants can efficiently discover the graph and optimally generate test data to accomplish test coverage.

Limitation:

Don't cope up the redundant states of the state chart model.

Number of states grows exponentially with the size of states in the state chart.

3.4.2 Li and Zhang et al. [21]

This paper presents a model of generating test data based on an improved ant colony optimization and path coverage criteria. In this paper, an approach combining the ant colony algorithm with the branch function technique to automatically generate test data based on path coverage criteria is proposed.

Limitation:

They didn't focus on the best proportion strategy to choose among poorest.

Character type problem are not handled in this techniques.

3.4.3 Huang et al. [33]

The purposed approach in this paper automatically generate GUI test cases. In ACO the generated test cases are feasible and optimized.

Reverse engineering framework is composed to create GUI structure and Event Flow Graph (EFG). ACO is used to generate test cases. The benefits of GUI test case generation using EFG are generation of model automatically which represent GUI objects to reduce the expenditure of complex modelling whereas test cases are executable. For gaining all the above goals automated test case generation through ACO is required.

Huang et al. presented an approach in which event flow model is used to achieve GUI object information through a new technique and implementing Ant Colony Optimization algorithm to find all possible event sequences.

In this Paper an Ant Colony Algorithm (ACO) is used for finding all optimal path in CFG of Software under test. This Algorithm is helpful for finding all Paths in between the nodes. Selection of path is depends on probability. The higher the probability means higher

chance of choosing that path. The probability value of path depends on Pheromone value and heuristic information of path. This is an effective approach which can easily generate optimal paths.

Limitation:

It only provide event coverage and don't provide full coverage of GUI like follow relation events (event interaction coverage).

A single test path cannot be used to detect all the possible defects in the software.

3.5 COVERAGE CRITERIA

Test case generation coverage criteria are interlinked. Most of the literature work exists on GUI test case generation that also focuses on describing the coverage criterion to achieve fault-free software.

3.5.1 Memon et al. [6]

Memon et al. explains various Coverage Criteria using event sequences to identify the adequacy for GUI software. Along with the adequacy of the software system the event sequences can be inaccurate due to the large amount of infeasible event. Memon's model in this paper also suffers inaccuracy. For example the three main events in this GUI component are YES, NO and CANCEL. Events NO and CANCEL are termination events because they terminate the modal window. However, the event YES can be a restricted-focus event or a termination event based on preconditions before the invocation of this modal dialog.

Limitation:

We cannot create an event flow graph for this component using Memon's definition because we cannot classify the YES event.

3.6 COMPARISON OF EXISTING TECHNIQUES

Table 3.1 shows the comparison of the existing techniques and methodologies used along with their limitations.

Author Name	Technique Used	Coverage Criteria	Single/Multiple path generation	Infeasible path possibility	Automated
Belli et al.	Finite state machines	State Interaction Sequences	Multiple	Yes	Partial
Memon	event-flow graph, Artificial intelligence	All event sequences	Not defined	Yes	Partial
White et al.	State machine, Complete interaction sequences	All paths	Multiple	Yes	Partial
Jin et al.	Finite State Machine , OP	State Transition	Not defined	Yes	Yes
Memon et al	Feedback	Event interaction sequences	Multiple	NO	Yes
Lu et al.	event-flow graph, ACO	All event sequences	Not defined	Yes	Yes
Memon et al.	event-interaction graph ,Feedback	Event interaction sequences	Multiple	NO	Partial
Khamis et al.	Genetic algorithm	Data coverage	Multiple	Yes	Yes
Rauf et al.	Genetic Algorithm	Event coverage	Multiple	Yes	Yes
Preeti et al.	state chart diagram, Genetic algorithm	All states sequence	Multiple	No	Yes
Li et al. 2007	State chart ,Ant Colony Optimization	All state coverage	Multiple	No	Partial
Li et al.2009	Ant Colony Optimization	All path coverage	Multiple	Yes	Yes
Huang et al.	Event flow graph & Ant Colony Optimization	All event coverage	Single	No	Yes

Table 3.1 : Comparison of different approaches

Chapter 4

Problem Definition

4 PROBLEM STATEMENT

Due to innovation in technology day by day GUI testing is very challenging for real time and safety critical system. GUI testing needs huge improvement to enhance the entire system's security, safety and reliability. GUI testing can be achieved either manually by software tester or automatically by automated methods.

GUI testing involves several tasks like to test all object events, mouse events, menus, fonts, images, content, control lists, etc. GUI testing is performed to check the user interface and test the functionality working properly or not. In GUI testing set of tasks are carried out to test the event sequences against the expected result. If the results differs with each other than it means there must be some faults in the software system.

To automate GUI testing some models are used in Model based Software testing (MBST). Traversing these models are easy and efficient. Simple graph traversal algorithm may generate feasible and infeasible test path. Infeasible paths are those paths which cannot execute by the given set of input.

Another challenging task is optimal test paths generation. To handle this problem ant colony optimization (ACO) algorithm is used. In ACO the generated test paths are optimized because path selection is depending upon the heuristic information. Heuristic value depend upon problem-oriented solution. Huang et al. uses ACO to generate optimized test paths which are not achieved through other simple traversing algorithm.

However the generated test paths only provide event coverage leaving some follow relation (child events) uncovered.

4.1 LIMITATION ON EXISTING METHODOLOGIES:

- Simple graph traversal algorithm can be applied for coverage but it can generate both feasible and infeasible paths. Infeasible paths cannot be implemented by any set of possible input values because the events are disabled or execution order among events are not appropriate.
- There are various coverage criterion used for adequate test data generation but event coverage criterion is widely used for this purpose. Event coverage criterion only cover events which are not enough for adequate testing.
- Infeasible test path generation.

Chapter 5

Proposed approach

5 PROPOSED APPROACH

Chapter 3 includes the literature survey of automated test data generation from state machine based model mainly FSM and event flow based model EFG. Nevertheless these models have some limitation also, because of complex and real nature of software. From literature we have identified the problems of infeasibility and single generated path which affect the overall performance of software system and sometimes produced flop software product. This chapter includes the proposed approach of event-flow graph (EFG) using Ant Colony Optimization algorithm for optimal test data generation.

Huang et al. [33] focused on the feasible test path generation from event flow graph along with the ant colony optimization algorithm for optimal data generation. In the paper of Huang a new framework based on user interface accessibility (UIA) is proposed. Ant colony algorithm (ACO) is used to generate feasible and optimal test data, which are useful for finding errors and faults by using EFG model.

Ant colony optimization algorithm generates test paths according to the probability of the events. Those event which have higher probability their chances of selection will be higher. The probability depends upon the pheromone value and the heuristic value, heuristic value tells the visibility of the event.

In proposed approach Ant Colony Optimization (ACO) which seeks for optimal test path generation. Optimal in term of efficiency and coverage which provides the coverage of all events and its edges based on probability and generating multiple paths from event-flow graph. The overall work includes the following steps:-

- To generate the test data, an approach is proposed that would satisfy all event coverage and event-pair interaction coverage criteria which provide all edges and all follow relation (child or adjacent) events in event flow graph.
- Construct an optimal solution that contains sequence of events to finish the

traversal of Event-Flow graph in the form of test cases.

- The Proposed approach ensures that each event and its edges in their traversal are executable at that time. This will ensure that only feasible paths are generated.
- Developed the strategy for feasible test data generation and how to cover all follow relationship events to eliminate the maximum faults from all perspectives
- To expand the test data generation technique the test paths are generated automatically and randomly.
- Comparing our results with existing approach
- Focus of the proposed approach is on feasible test paths generation and provide full coverage

5.1 RESEARCH APPROACH

The proposed research approach contains coverage of Event-Flow graph and optimal test path generation which will explain in detail in this section.

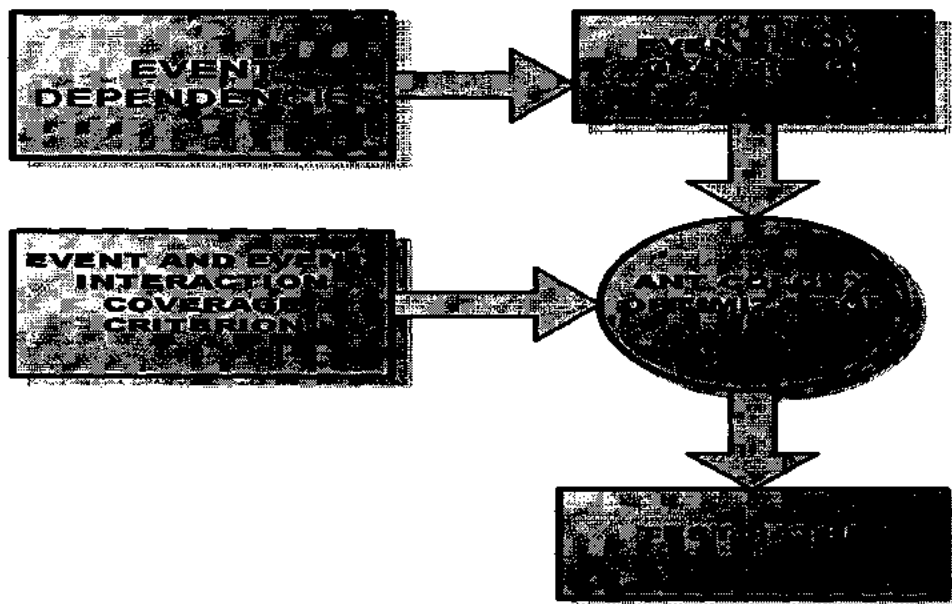


Fig 5.1 Abstract level Research Approach

First of all input is given in the form of Event-flow graph (EFG) which tells events and their dependencies in the form of edges. In the proposed approach a program (ACO) is

selected under test that randomly selects the initial event for test paths generation. In ACO, the probability of each event is calculated by using the formula in (Huang et al.) After calculating the probability we have to choose one node having the largest probability among all of them. Another input is given to ant colony optimization (ACO) algorithm is coverage criterion which guarantees that each event and each edges are visited. At the end the generated output is in the form of test path which shows the sequence of events and edges.

5.2 ANT COLONY OPTIMIZATION TECHNIQUE

Ant Colony Optimization (ACO) is a Meta-heuristic method to solve combinatorial optimization problems by using behavior of ant colonies in real. In ACO algorithms numerous generation of ants search for upright solutions. Each ant builds a solution step by step going through a number of probabilistic decisions until a solution is established. Frequently, ants that found a good solution mark their paths by depositing the amount of pheromone on the edges. After that ants of the next generation are attracted by the pheromone which was deposited by previous ants so that they will search improved solutions. In reality ants are expert of finding the shortest path from a source to destination in search of food (Li, Zhang et al. 2009)

$$\text{Probability calculation formula: } p_{xy} = \frac{(\tau_{xy\alpha})(\eta_{xy\beta})}{\sum (\tau_{xy\alpha})(\eta_{xy\beta})} \quad (1)$$

Description about ACO parameters:

Pheromone trail are represented by tau (τ) showing the pheromone amount from one node to another node and is being continuously updated as the paths are navigated. Heuristic information is represented by eta (η) showing the attractiveness of the path as shown in equation (1). x is the initial event and destination event is denoted by y. This equation is used to calculate the probability for the ant to choose an accurate path. Probability depends on the feasible set, heuristic and the pheromone level of the

corresponding path. Alpha (α) determines the relative importance of pheromone value and beta (β) defines the visibility of heuristic information. Evaporation rate (ρ) must be greater than 0 and less than 1. These are tuning parameter which are used for selecting the optimal test paths and feasible solution.

5.2.1 Parameter setting

Parameter setting for ACO are shown in table 5.1

Parameter	Value
INITIAL PHEROMONE VALUE (τ)	0.2
ALPHA (α)	0.2
BETA (β)	0.9
PHEROMONE EVAPORATION RATE (ρ)	0.3

Table 5.1 ACO Parameter Setting

Formerly when all ants have finished their tour the amount of pheromone on trail (path) is restructured (by using the global updating rule).

Pheromone Updating rule: $\tau_{xy}(t) \leftarrow (1 - \rho) \cdot \tau_{xy}(t) + \rho \cdot \tau_0$ (where ρ is the evaporation rate) [21] [33]

The pheromone updating rule is calculated so that they have a tendency to give more pheromone toward edges which must be visited by ants. In general, the higher the two values, the higher the probability of choosing the linked edge. Updating the pheromone trail values in two phases. First, pheromone evaporation is applied to decrease pheromone values. The aim of pheromone evaporation is to avoid an infinite increase of pheromone values and to allow the ant colony to manage poor choices done previously. Pheromone deposit is applied to increase the pheromone values that fit to good solutions the ants have engendered.

5.3 PROPOSED APPROACH

The description of the proposed approach in detail is given below which describe every steps of Fig 5.2

1. **Build Event Flow Graph(EFG) model**

Firstly analyze the entire requirement carefully then identify all the events regarding functionality and their dependency with each other. When all events are identified model them in Event Flow Graph (EFG) which represents the events and their relationship with other events.

2. **Ant Colony Optimization Algorithm**

a) **Put Ant on EFG**

For GUI events traversing place the ants on the Event Flow Graph. These artificial ants would respond as natural ant practice for food search. Ants seeks for the optimal solution.

b) **Ant records the number of nodes**

Input of Event flow graph is given in the form of Xml. Then ant traverses the graph and records the total number of nodes and its adjacent events (child event).

c) **Traverse the initial event**

After calculating total number of nodes and its adjacent nodes the ants traverse the initial node randomly.

d) **Calculate probability**

Calculate the probability of adjacent events of the current event. Choose the highest probability node among all adjacent node of current event.

e) **Update local pheromone value**

After the event of high probability is selected. Update the pheromone value using formula which is given above pp. This is step by step updating of pheromone value.

f) **Traverse until end node**

Traverse or visit all events and event-pair interaction (follow-relation events) through calculating the probability until the initial event achieved.

g) **Record the path**

Record the path of all node which have highest probability among other child or adjacent nodes. Record each event step by step having greatest probability.

h) **Print generated path**

Print the recorded path which shows the all the events and their path towards other events depending upon the probability and covers unvisited events on high priority.

i) **Update global pheromone**

When the path has been recorded. Update the global pheromone value using formula. The aim of pheromone evaporation is to avoid an infinite increase of pheromone values and to allow the ant colony to manage poor choices done previously.

3. **Coverage criterion fulfilled**

After all the paths are traversed check whether the event coverage and event interaction coverage criterion satisfied or not. If the paths covers all events and their edges its means that coverage criterion is complete otherwise repeat the procedure of event interaction coverage using ACO rule.

4. **Print all paths having events and event- pair relation**

At the end when coverage criterion is fully satisfied. Print all the paths having events and event interaction nodes. The generated test paths shows the path having sequence of events and edges which shows event-pair relation.

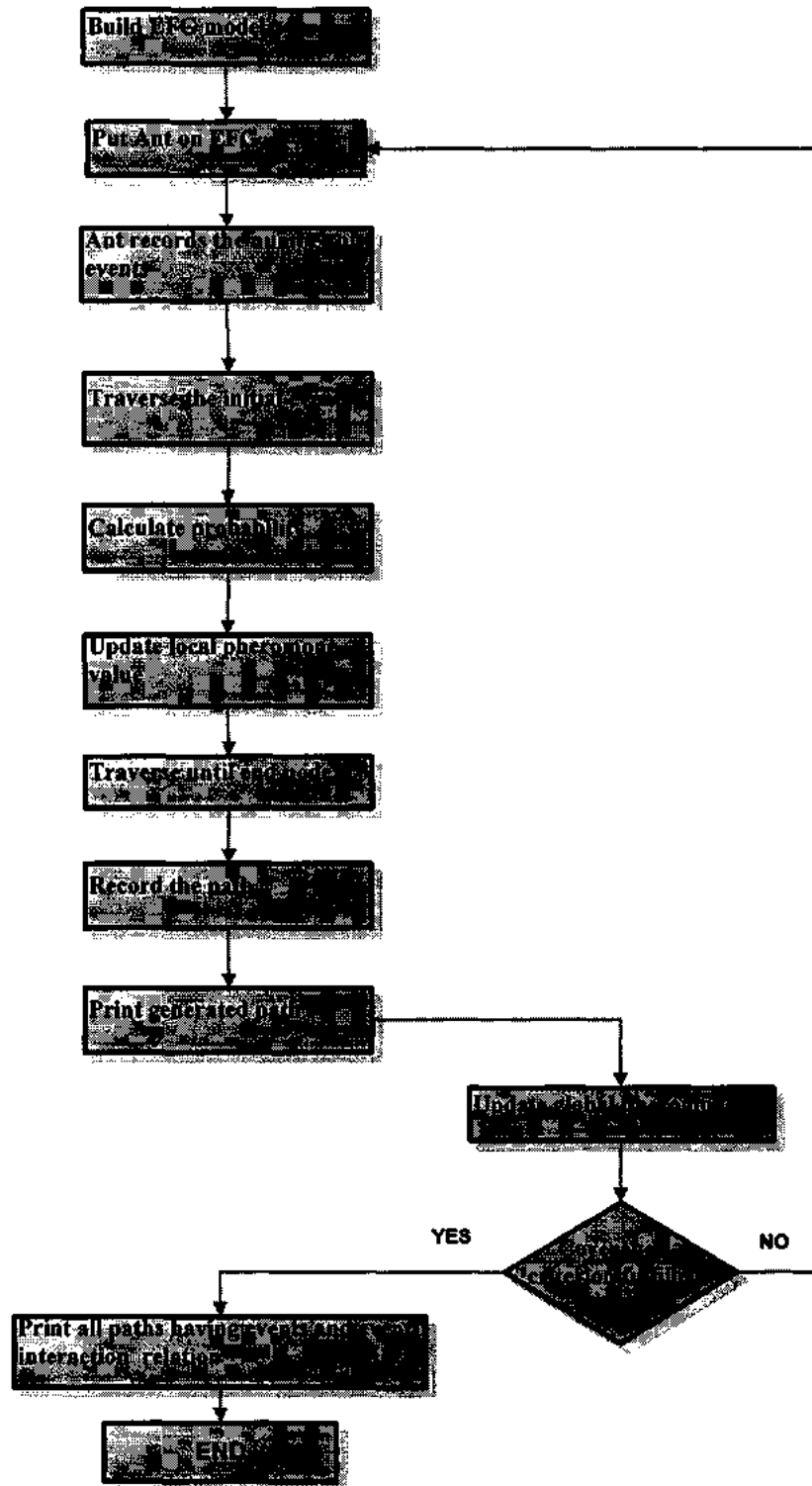


Fig 5.2: Diagram of Proposed Approach

5.4 PROPOSED APPROACH ALGORITHM

1. Initialization:

- a. Set initial parameters: variable, events, function, input trail and output path.
- b. Set initial pheromone value, pheromone evaporation rate, alpha, beta and individual pheromone rate.
- c. Start traversing the node from the initial event.

◆ While termination conditions do not encounter, do (visit all events)

2. Traverse the graph using TraverseGraph function

◆ While termination conditions do not meet, do

a. Construct Optimal Solution:

Each ant constructs a path by continuously calculating the `getHighestProbability` function which tells the probability of adjacent event depend upon the attractiveness of the path, and the pheromone level of the path.

1. Select highest probability event
2. Apply Local pheromone updating rule
3. Record the path (trail)

End While

3. If the path is traversed, update it.

- a) Update Trails
- b) Update global updating rule (it contains pheromone deposit and pheromone evaporation)

- Evaporate pheromone on a fixed amount continuously (which is less than 1 and greater than 0).
- For each trail apply global pheromone update.
- Emphasize the best tour by depositing the Individual pheromone on the trail.

4. ♦ while coverage criterion full filled

Print event and edges path

Else

Go to step 2 (traverse the graph)

End While

EndWhile

5.5 METHODOLOGY USED TO PROVE

To test our proposed methodology following test paths are generated. These results are generated from 1st iteration. The program first ask from the user which types of paths are generated. Two option given to the user either to select event coverage or select edge coverage which shows the edges between events. The results are saved into the file which contains all the event coverage and all edge coverage.

Event Path 1: filemenuitem -> openmenuitem -> filetypeitem -> encodeitem -> filenameedit -> encodeitem -> filetypeitem -> filenameedit -> cancelopen

Edge Path 1 : (filemenuitem-openmenuitem) -> (openmenuitem-filetypeitem) -> (filetypeitem-filenameedit) -> (filenameedit-filetypeitem) -> (filetypeitem-encodeitem) -> (encodeitem-filetypeitem) -> (filetypeitem-filenameedit) -> (filenameedit-encodeitem) -> (encodeitem-filenameedit) -> (filenameedit-filetypeitem) -> (filetypeitem-encodeitem) -> (encodeitem-filetypeitem) -> (filetypeitem-openbutton) -> (openbutton-filemenuitem)

Chapter 6

Implementation

6 IMPLEMENTATION

In this chapter we will describe the implementation phases of proposed approach. It also includes the implementation of Ant Colony Optimization for optimized test path generation along with the feasible test paths generation and then presents the results of proposed approach.

6.1 TOOL ARCHITECTURE

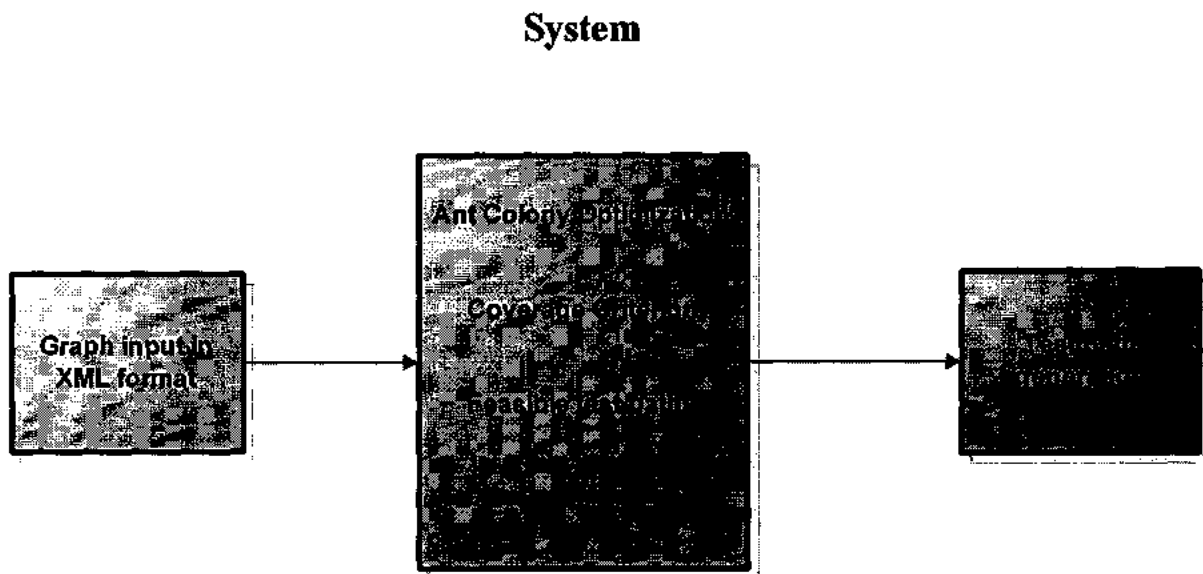


Fig 6.1 Tool Architecture

6.2 DESCRIPTION ABOUT TOOL ARCHITECTURE

The detail of proposed approach tool is as follows:

1. Graph Input

Input of a program is event flow graph (EFG). Graph is saved in the XML file and then used by system under test (SUT). In XML file the event is given with their child

nodes for easily access.

2. Ant Colony Optimization

Ant colony Optimization algorithm is used for optimal generation of test path. ACO algorithm depend upon the pheromone value and the heuristic value. Heuristic value depend upon the problem-oriented solution. Here problem-oriented solution is to cover all events and their edges between events.

3. Coverage Criterion

The main purpose of my research is to answer that how much testing is required for fault free system. For testing we need coverage of full system so that maximum error would be detected. There are many types of coverage criterion. In previous technique event coverage criterion is used that is not enough to produce error free system. For accurate and adequate system we purposed event pair coverage criterion that ensure that each event and each edge between events is covered at least once.

4. Feasible test paths

In simple graph traversal algorithm there are chance of feasible and infeasible test paths generation which leads to the state which are not available at that time. This problem lead to number of useless test paths.

5. Test path generation

The resulted output is in the form of generated test paths which covers all the events and the edges between events. Also the generated test paths are feasible test paths.

6.3 GRAPH REPRESENTATION

The above graph example is saved in XML format which can be easily accessed by the program. The xml format shows the nodes of EFG and its child or adjacent nodes.

<Graph>

<Node name="File_Menu_Item">

<ChildNode name="Open_Menu_Item"/>


```
        <ChildNode name="Edit_Menu_Item"/>
</Node>
<Node name="Open_Menu_Item">
    <ChildNode name="File_Menu_Item"/>
    <ChildNode name="Encode_Item"/>
    <ChildNode name="File_Name_Edit"/>
    <ChildNode name="File_Type_Item"/>
    <ChildNode name="Open_Button "/>
    <ChildNode name="Cancle_Open"/>
</Node>
<Node name="Encode_Item">
    <ChildNode name="File_Name_Edit"/>
    <ChildNode name="File_Type_Item"/>
    <ChildNode name="Open_Button "/>
    <ChildNode name="Cancle_Open"/>
</Node>
<Node name="File_Name_Edit">
    <ChildNode name="File_Type_Item"/>
    <ChildNode name="Open_Button"/>
    <ChildNode name="Cancle_Open"/>
    <ChildNode name="Encode_Item"/>
</Node>
<Node name="File_Type_Item">
    <ChildNode name="File_Name_Edit"/>
    <ChildNode name="Encode_Item"/>
    <ChildNode name="Open_Button"/>
    <ChildNode name="Cancle_Open"/>

```

```
</Node>
<Node name="Cancle_Open">
  <ChildNode name="File_Menu_Item"/>
</Node>
<Node name="Open_Button">
  <ChildNode name="File_Menu_Item"/>
</Node>
<Node name="Edit_Menu_Item">
  <ChildNode name="File_Menu_Item"/>
  <ChildNode name="Copy_Menu_Item"/>
  <ChildNode name="Paste_Menu_Item"/>
  <ChildNode name="Find_Menu_Item"/>
</Node>
<Node name="Copy_Menu_Item">
  <ChildNode name="Edit_Menu_Item"/>
</Node>
<Node name="Paste_Menu_Item" depends_on="Copy_Menu_Item">
  <ChildNode name="Copy_Menu_Item"/>
  <ChildNode name="Edit_Menu_Item"/>
</Node>
<Node name="Find_Menu_Item">
  <ChildNode name="Edit_Menu_Item"/>
  <ChildNode name="Down_Item"/>
  <ChildNode name="Up_Item"/>
  <ChildNode name="Find_Edit"/>
  <ChildNode name="Find_Button"/>
  <ChildNode name="Cancle_Find"/>
```

```
        <ChildNode name="Case_Check"/>
    </Node>
    <Node name="Down_Item">
        <ChildNode name="Edit_Menu_Item"/>
        <ChildNode name="Up_Item"/>
        <ChildNode name="Case_Check"/>
        <ChildNode name="Find_Edit"/>
        <ChildNode name="Find_Button"/>
        <ChildNode name="Cancle_Find"/>
    </Node>
    <Node name="Up_Item">
        <ChildNode name="Edit_Menu_Item"/>
        <ChildNode name="Down_Item"/>
        <ChildNode name="Case_Check"/>
        <ChildNode name="Find_Edit"/>
        <ChildNode name="Find_Button"/>
        <ChildNode name="Cancle_Find"/>
    </Node>
    <Node name="Case_Check">
        <ChildNode name="Edit_Menu_Item"/>
        <ChildNode name="Down_Item"/>
        <ChildNode name="Up_Item"/>
        <ChildNode name="Find_Edit"/>
        <ChildNode name="Find_Button"/>
        <ChildNode name="Cancle_Find"/>
    </Node>
    <Node name="Find_Edit">
```

```
<ChildNode name="Edit_Menu_Item"/>
<ChildNode name="Down_Item"/>
<ChildNode name="Up_Item"/>
<ChildNode name="Case_Check"/>
<ChildNode name="Find_Button"/>
<ChildNode name="Cancle_Find"/>
</Node>
<Node name="Find_Button" depends_on=Find_Edit">
    <ChildNode name="Edit_Menu_Item"/>
</Node>
<Node name="Cancle_Find">
    <ChildNode name="Edit_Menu_Item"/>
</Node>
</Graph>
```

6.4 IMPLEMENTATION DETAILS

Step 1: Initialization

```
Double ALPHA, Double BETA
Double PHEROMONE_EVAPORATION_RATE
Double INITIAL_PHEROMONE_VALUE
Double INDIVIDUAL_PHEROMONE_VALUE
```

Step 2: Start traversing the edge from the initial event

```
while (hasUnVisitedEdges()) {  
    if (selectedEdgeLabel != null && !selectedEdgeLabel.isEmpty()) {  
        edgeIndex.setVisited(true);  
        path.append(selectedEdgeLabel);  
        nodeLabel = tempEdge.getEndingNode();    }}
```

Step 3: Construct Optimal Solution:

```
for (String childNode : childNodes) {  
    int nodeIndex = this.nodeIndexMap.get(childNode);  
    Node tempNode = this.nodes.get(nodeIndex);  
    double updatedPheromoneValue = ((1 -  
Constants.PHEROMONE_EVAPORATION_RATE) *  
(Constants.INDIVIDUAL_PHEROMONE_VALUE));    }
```

A. Select highest probability edge

```
for (String nodeEdge : nodeEdgesList) {
    String dependsOn = tempEdge.getDependsOn();
    if (dependsOn != null && !dependsOn.isEmpty()) {
        Edge dependsOnEdge = this.edges.get(this.edgeIndexMap.get(dependsOn));
        if (dependsOnEdge.isVisited()) {
            double occurrence = calculateOccurrenceOfEdge(tempEdge);
            tempEdge.setOccurrence(occurrence);
            totalOccurrences += occurrence; }
    else { tempEdge.setOccurrence(0.0); } } else {
        double occurrence = calculateOccurrenceOfEdge(tempEdge);
        tempEdge.setOccurrence(occurrence);
        totalOccurrences += occurrence; } }

for (String nodeEdge : nodeEdgesList) {
    double probability = tempEdge.getOccurrence() / totalOccurrences;
    probabilities.add(probability);
    probabilityEdgeMap.put(probability, nodeEdge); }

Collections.sort(probabilities);
Collections.reverse(probabilities);
highestProbabilityEdge = probabilityEdgeMap.get(probabilities.get(0));
return highestProbabilityEdge;
```

B. Apply Local pheromone updating rule

C. Record the path (trail)

```
String selectedEdge = getHigestProbabilityEdge(selectedNode);
    tempEdge = this.edges.get(this.edgeIndexMap.get(selectedEdge));
    evaporateEdgePheromoneValue(tempEdge);
    pathEdges.add(selectedEdgeLabel);
    } while (!nodeLabel.equals(this.rootNode.getLabel()));
updateEdgePathPheromoneValue(pathEdges);
System.out.println("Edge Path: " + path.toString());
```

Step 4: Update global updating rule (it contains pheromone deposit and pheromone evaporation)

```
for (String tempEdgeLabel : pathEdges) {
double evaporatedPheromoneValue = ((1 - Constants.PHEROMONE_EVAPORATION_RATE) *
edge.getPheromoneValue()) + (Constants.PHEROMONE_EVAPORATION_RATE *
Constants.INITIAL_PHEROMONE_VALUE);
    edge.setPheromoneValue(evaporatedPheromoneValue); } }
```

Step 5. End

6.5 PROPOSED FRAMEWORK INTERFACE

The Graphical User Interface (GUI) of the proposed methodology is shown in Fig. 6.2

First of all we have to select the input file in the form of XML. Another option Output Directory saves the result of generated paths of nodes and edges in the selected location or directory. Than the results are generated upon the selection of various choices like

traverse nodes, traverse edges, generate nodes output file and generate edges output file.



Fig 6.2 GUI of proposed framework

Chapter 7

Case Studies

7 CASE STUDIES

To justify our proposed approach, four case studies are used to perform experiment. One on them is the example of Notepad Graphical User Interface (GUI) which is already used in existing paper [33]. Example shown below in Fig 7.1 demonstrate the events and edges. Events define the functionality of the system represented in circle and the arrow between them shows the dependency between each other.

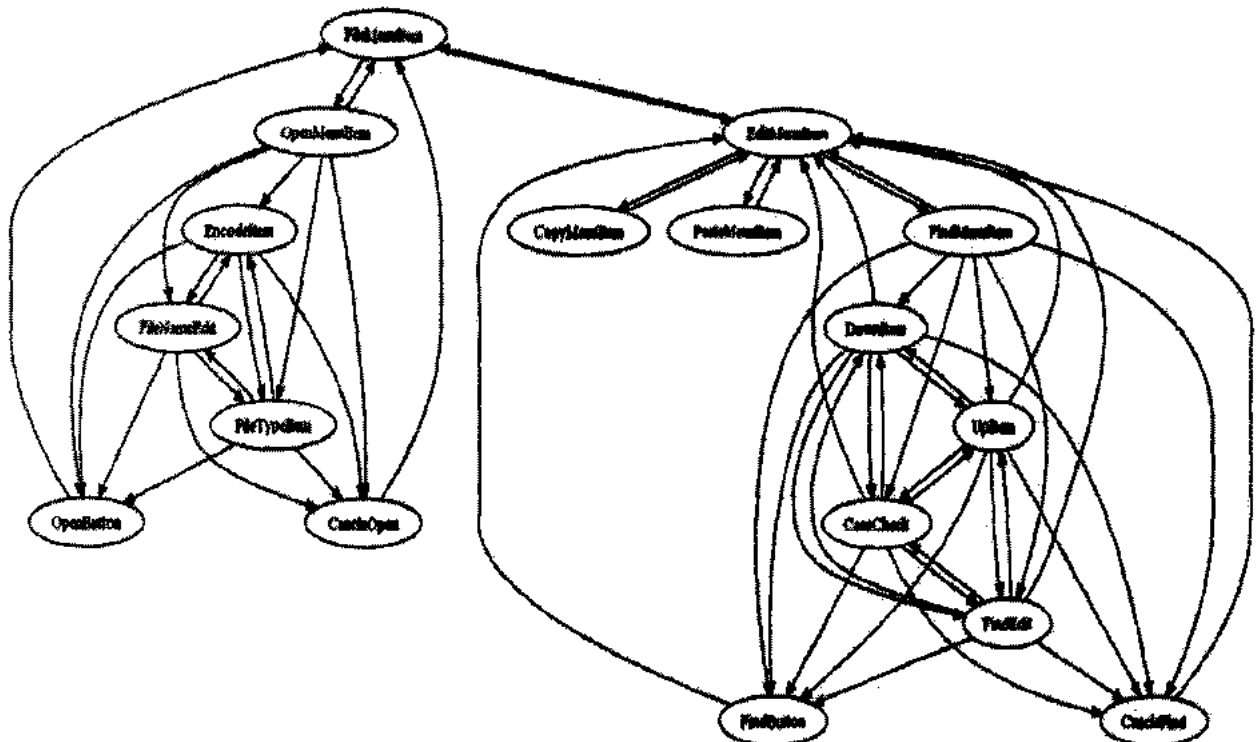


Fig 7.1 Event Flow Graph (EFG) Example of Notepad [33]

7.1 DESCRIPTION ABOUT EXAMPLE

We are using the same example of Notepad for proposed approach as used [33]. This example expresses how dynamically the ants create the feasible test cases. At first choose the initial event randomly which is not depending on other events. Then at every step it travels along with the follow relationship events (adjacent events) of the EFG. The program chooses an event by calculating the probability of each adjacent event and leave

some pheromone on that selected adjacent event. The probabilistic rule is based on pheromone rate and heuristic information. The probability will be higher when the pheromone and the heuristic value of an event will be higher after that an ant will choose that particular event.

$$\text{Probability calculation formula: } p_{xy} = \frac{(\tau_{xy}^\alpha)(\eta_{xy}^\beta)}{\sum (\tau_{xy}^\alpha)(\eta_{xy}^\beta)} \quad (1)$$

τ is the current pheromone rate placed on every node and η is the heuristic information which is calculated by sum of followed by events of current event(y) +1 / sum of event y have been visited +1 as shown in equation 1. Choose the highest probability node after that update pheromone rate on that chosen event.

$$\text{Local pheromone updation rule: } \tau_{xy}(t) \leftarrow (1 - \rho) \cdot \tau_{xy} \quad (2)$$

After every step the pheromone values will be decreased by fixed amount by applying local pheromone updating rule. An ant created a solution when it has visited all the events and edges or it cannot move onward (no follow events left unvisited). $\rho \in (0, 1]$ is the evaporation rate shown in equation 2.

Take the EFG of notepad as shown in Fig. 7.1 at starting an initial event (filemenuitem) is selected randomly from Event Flow Graph filemenuitem is in enabled mode means it is not dependent upon other event. editmenuitem and openmenuitem are the adjacent event of filemenuitem which shows follow relation. To choose the next event, probability of current event is calculated by equation (1) (filemenuitem) follow relations e.g. editmenuitem, openmenuitem. τ (tau) = 0.2, η (eta) of openmenuitem = 6+1 / 0+1 = 7, η of editmenuitem = 4+1 / 0+1 = 5

$$P(\text{filemenuitem, openmenuitem}) = \frac{0.2^{0.2} * 7^{0.9}}{0.2^{0.2} * 7^{0.9} + 0.2^{0.2} * 5^{0.9}} = 0.5751$$

$$P(\text{filemenuitem, editmenuitem}) = \frac{0.2^{0.2} * 5^{0.9}}{0.2^{0.2} * 5^{0.9} + 0.2^{0.2} * 7^{0.9}} = 0.4248$$

The probability of openmenuitem is greater than probability of editmenuitem, so we choose openmenuitem. After that update pheromone value as shown in equation (2).

$T_{(\text{openmenuitem})} \leftarrow (1 - 0.3) \cdot 0.2 = 0.14$ so current pheromone value is decreased by 0.14.

Now we check the follow relation of openmenuitem for choosing the next event through the probability calculation formula and local pheromone updating rule is applied on the selected event. Similarly do until all events are traversed.

When a path is traversed deposit some individual pheromone to strengthen the optimized path and pheromone evaporation to remove the infinity of the increase in pheromone value as shown below in equation (3)

Global pheromone updation rule: $T_{(y)} \leftarrow (1 - \rho) \cdot T_{xy}(t) + \rho \cdot T_0$ (3)

ρ is the pheromone evaporation rate . T_0 is the initial pheromone value and T_{xy} is the current value of pheromone at time t .

When one path is generated or traversed update the global pheromone value on the whole path. On every iteration the generated test path generates the following sequences. The output of 1st iteration is: filemenuitem -> openmenuitem -> filetypeitem -> encodeitem -> filenameedit -> encodeitem -> filetypeitem -> filenameedit -> cancelopen

When one path is generated the control goes to the initial node then start traversing the path again similarly in each iteration the program generate the sequence of events in the test paths and edges.

7.2 GENERATED TEST PATHS

All traversed test path events are following which shows that every event is covered or visited at least once:

Path 1: filemenuitem -> openmenuitem -> filetypeitem -> encodeitem -> filenameedit -> encodeitem -> filetypeitem -> filenameedit -> cancelopen

Path 2: filemenuitem -> editmenuitem -> findmenuitem -> casecheck -> findedit -> upitem -> downitem -> findedit -> casecheck -> upitem -> downitem -> editmenuitem -> findmenuitem -> cancelfind -> editmenuitem -> findmenuitem -> findbutton -> editmenuitem -> copymenuitem -> editmenuitem -> pastemenuitem -> copymenuitem -> editmenuitem -> findmenuitem -> casecheck -> findedit -> upitem -> downitem -> findedit -> casecheck -> upitem -> downitem -> cancelfind -> editmenuitem -> pastemenuitem -> copymenuitem -> editmenuitem -> findmenuitem -> findbutton -> editmenuitem -> findmenuitem -> casecheck -> findedit -> upitem -> downitem -> findedit -> casecheck -> upitem -> downitem -> cancelfind -> editmenuitem -> pastemenuitem -> copymenuitem -> editmenuitem

Path 3: filemenuitem -> openmenuitem -> openbutton

No of path generation	Event coverage in Existing approach	Event coverage in Proposed approach
Path 1	17	6
Path 2	0	10
Path 3	0	1
Total number of events	17	17

Table 7.1 Comparison of event coverage

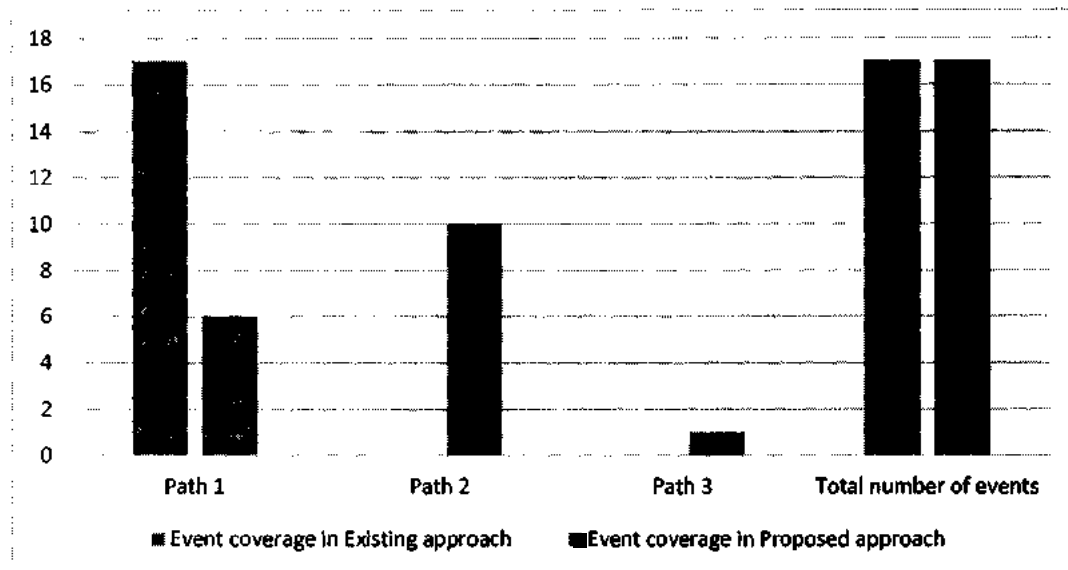


Fig. 7.2 Comparison of event coverage in existing and proposed approach

All traversed edges of event flow graph which covers all event-pair relation are as follows:

Edge Path:

Edge Path 1: (filemenuitem-openmenuitem) -> (openmenuitem-filetypeitem) -> (filetypeitem-filenameedit) -> (filenameedit-filetypeitem) -> (filetypeitem-encodeitem) ->

(encodeitem-filetypeitem) -> (filetypeitem-filenameedit) -> (filenameedit-encodeitem) -> (encodeitem-filenameedit) -> (filenameedit-filetypeitem) -> (filetypeitem-encodeitem) -> (encodeitem-filetypeitem) -> (filetypeitem-openbutton) -> (openbutton-filemenuitem)

Edge Path 2: (filemenuitem-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-findedit) -> (findedit-casecheck) -> (casecheck-findedit) -> (findedit-upitem) -> (upitem-findedit) -> (findedit-downitem) -> (downitem-findedit) -> (findedit-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-casecheck) -> (casecheck-upitem) -> (upitem-casecheck) -> (casecheck-downitem) -> (downitem-casecheck) -> (casecheck-editmenuitem) -> (editmenuitem-pastemenuitem) -> (pastemenuitem-editmenuitem) -> (editmenuitem-filemenuitem)

Edge Path 3: (filemenuitem-openmenuitem) -> (openmenuitem-filenameedit) -> (filenameedit-encodeitem) -> (encodeitem-filenameedit) -> (filenameedit-openbutton) -> (openbutton-filemenuitem)

Edge Path 4: (filemenuitem-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-upitem) -> (upitem-downitem) -> (downitem-upitem) -> (upitem-editmenuitem) -> (editmenuitem-copymenuitem) -> (copymenuitem-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-downitem) -> (downitem-editmenuitem) -> (editmenuitem-pastemenuitem) -> (pastemenuitem-editmenuitem) -> (editmenuitem-filemenuitem)

Edge Path 5: (filemenuitem-openmenuitem) -> (openmenuitem-encodeitem) -> (encodeitem-openbutton) -> (openbutton-filemenuitem)

Edge Path 6: (filemenuitem-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-editmenuitem) -> (editmenuitem-copymenuitem) -> (copymenuitem-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-findedit) -> (findedit-casecheck) -> (casecheck-findedit) -> (findedit-upitem) -> (upitem-findedit) -> (findedit-downitem) -> (downitem-findedit) -> (findedit-editmenuitem) -> (editmenuitem-pastemenuitem) -> (pastemenuitem-copymenuitem) -> (copymenuitem-editmenuitem) -> (editmenuitem-filemenuitem)

Edge Path 7: (filemenuitem-openmenuitem) -> (openmenuitem-filemenuitem)

Edge Path 8: (filemenuitem-openmenuitem) -> (openmenuitem-filetypeitem) -> (filetypeitem-candleopen) -> (candleopen-filemenuitem)

Edge Path 9: (filemenuitem-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-casecheck) -> (casecheck-upitem) -> (upitem-casecheck) -> (casecheck-downitem) -> (downitem-casecheck) -> (casecheck-editmenuitem) -> (editmenuitem-pastemenuitem) -> (pastemenuitem-editmenuitem) -> (editmenuitem-filemenuitem)

Edge Path 10: (filemenuitem-openmenuitem) -> (openmenuitem-filenameedit) -> (filenameedit-candleopen) -> (candleopen-filemenuitem)

Edge Path 11: (filemenuitem-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-upitem) -> (upitem-downitem) -> (downitem-upitem) -> (upitem-editmenuitem) -> (editmenuitem-copymenuitem) -> (copymenuitem-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-downitem) -> (downitem-editmenuitem) -> (editmenuitem-pastemenuitem) -> (pastemenuitem-editmenuitem) -> (editmenuitem-filemenuitem)

Edge Path 12: (filemenuitem-openmenuitem) -> (openmenuitem-encodeitem) -> (encodeitem-cancleopen) -> (cancleopen-filemenuitem)

Edge Path 13: (filemenuitem-editmenuitem) -> (editmenuitem-copymenuitem) -> (copymenuitem-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-editmenuitem) -> (editmenuitem-pastemenuitem) -> (pastemenuitem-copymenuitem) -> (copymenuitem-editmenuitem) -> (editmenuitem-filemenuitem)

Edge Path 14: (filemenuitem-openmenuitem) -> (openmenuitem-openbutton) -> (openbutton-filemenuitem)

Edge Path 15: (filemenuitem-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-canclefind) -> (canclefind-editmenuitem) -> (editmenuitem-copymenuitem) -> (copymenuitem-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-findbutton) -> (findbutton-editmenuitem) -> (editmenuitem-pastemenuitem) -> (pastemenuitem-editmenuitem) -> (editmenuitem-filemenuitem)

Edge Path 16: (filemenuitem-openmenuitem) -> (openmenuitem-cancleopen) -> (cancleopen-filemenuitem)

Edge Path 17: (filemenuitem-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-findedit) -> (findedit-canclefind) -> (canclefind-editmenuitem) -> (editmenuitem-copymenuitem) -> (copymenuitem-editmenuitem) -> (editmenuitem-pastemenuitem) -> (pastemenuitem-copymenuitem) -> (copymenuitem-editmenuitem) -> (editmenuitem-filemenuitem)

Edge Path 18: (filemenuitem-openmenuitem) -> (openmenuitem-filetypeitem) -> (filetypeitem-filenameedit) -> (filenameedit-filetypeitem) -> (filetypeitem-encodeitem) -> (encodeitem-filetypeitem) -> (filetypeitem-filenameedit) -> (filenameedit-encodeitem) -> (encodeitem-filenameedit) -> (filenameedit-filetypeitem) -> (filetypeitem-encodeitem) -> (encodeitem-filetypeitem) -> (filetypeitem-openbutton) -> (openbutton-filemenuitem)

Edge Path 19: (filemenuitem-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-casecheck) -> (casecheck-canclefind) -> (canclefind-editmenuitem) -> (editmenuitem-copymenuitem) -> (copymenuitem-editmenuitem) -> (editmenuitem-pastemenuitem) -> (pastemenuitem-editmenuitem) -> (editmenuitem-filemenuitem)

Edge Path 20: (filemenuitem-openmenuitem) -> (openmenuitem-filenameedit) -> (filenameedit-encodeitem) -> (encodeitem-filenameedit) -> (filenameedit-openbutton) -> (openbutton-filemenuitem)

Edge Path 21: (filemenuitem-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-upitem) -> (upitem-canclefind) -> (canclefind-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-downitem) -> (downitem-canclefind) -> (canclefind-editmenuitem) -> (editmenuitem-pastemenuitem) -> (pastemenuitem-editmenuitem) -> (editmenuitem-filemenuitem)

Edge Path 22: (filemenuitem-openmenuitem) -> (openmenuitem-encodeitem) -> (encodeitem-openbutton) -> (openbutton-filemenuitem)

Edge Path 23: (filemenuitem-openmenuitem) -> (openmenuitem-filemenuitem)

Edge Path 24: (filemenuitem-editmenuitem) -> (editmenuitem-copymenuitem) -> (copymenuitem-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-editmenuitem) -> (editmenuitem-pastemenuitem) -> (pastemenuitem-copymenuitem) -> (copymenuitem-editmenuitem) -> (editmenuitem-filemenuitem)

Edge Path 25: (filemenuitem-openmenuitem) -> (openmenuitem-filetypeitem) -> (filetypeitem-cancleopen) -> (cancleopen-filemenuitem)

Edge Path 26: (filemenuitem-editmenuitem) -> (editmenuitem-copymenuitem) -> (copymenuitem-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-findedit) -> (findedit-findbutton) -> (findbutton-editmenuitem) -> (editmenuitem-pastemenuitem) -> (pastemenuitem-editmenuitem) -> (editmenuitem-filemenuitem)

Edge Path 27: (filemenuitem-openmenuitem) -> (openmenuitem-filenameedit) -> (filenameedit-cancleopen) -> (cancleopen-filemenuitem)

Edge Path 28: (filemenuitem-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-casecheck) -> (casecheck-findbutton) -> (findbutton-editmenuitem) -> (editmenuitem-copymenuitem) -> (copymenuitem-editmenuitem) -> (editmenuitem-pastemenuitem) -> (pastemenuitem-editmenuitem) -> (editmenuitem-filemenuitem)

Edge Path 29: (filemenuitem-openmenuitem) -> (openmenuitem-encodeitem) -> (encodeitem-cancleopen) -> (cancleopen-filemenuitem)

Edge Path 30: (filemenuitem-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-upitem) -> (upitem-findbutton) -> (findbutton-editmenuitem) -> (editmenuitem-copymenuitem) -> (copymenuitem-editmenuitem) -> (editmenuitem-findmenuitem) -> (findmenuitem-downitem) -> (downitem-findbutton) -> (findbutton-editmenuitem) -> (editmenuitem-pastemenuitem) -> (pastemenuitem-copymenuitem) -> (copymenuitem-editmenuitem) -> (editmenuitem-filemenuitem)

No of path generation	Unvisited edge coverage in Existing approach	Unvisited edge coverage in Proposed approach
Path1	35	11
Path2	0	19

Path3	0	2
Path4	0	8
Path 5	0	2
Path 6	0	1
Path 7	0	1
Path 8	0	2
Path 9	0	1
Path 10	0	1
Path 11	0	1
Path 12	0	4
Path 13	0	1
Path 14	0	1
Path 15	0	2
Path 16	0	1
Path 17	0	1
Path 18	0	2
Total number of edges covered	35	61

Table 7.2 Comparison of Event-interaction Coverage

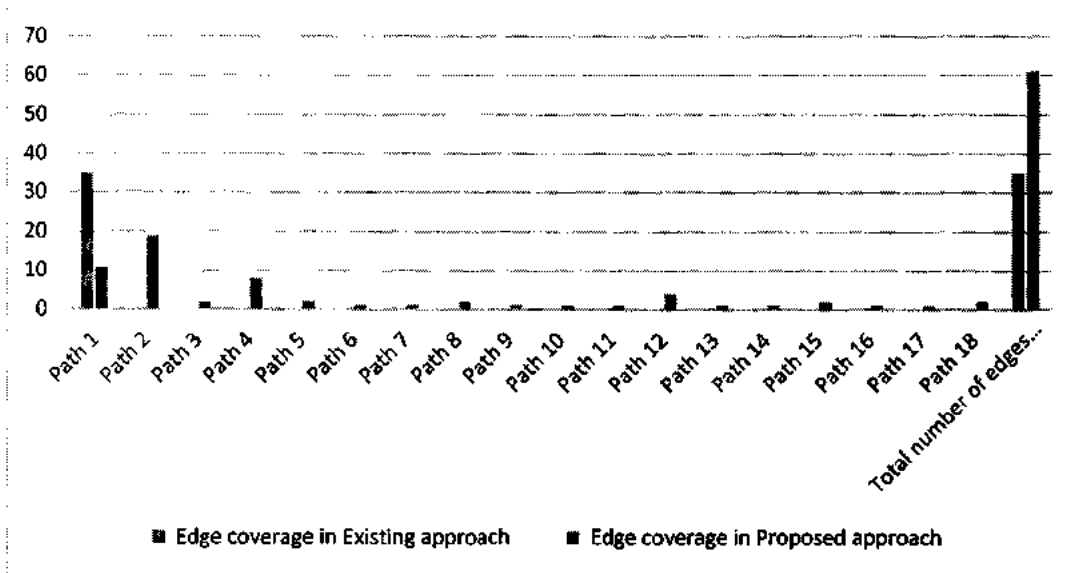


Fig. 7.3 Comparison of edge coverage in existing and proposed approach

From the table and the generated paths it is proved that the proposed approach provide better coverage than the existing approach. The proposed approach generates the all events and all edges which enhance the testing of GUI system.

7.3 CASE STUDY 2

There is another example which strengthens the proposed approach. Internet Explorer (formerly Microsoft Internet Explorer) abbreviated as IE is a series of graphical web browsers. Internet Explorer is a web based Graphical user interface which provides ease to the browsing all over the world. Internet Explorer (IE) is one of the most widely used web browsers.

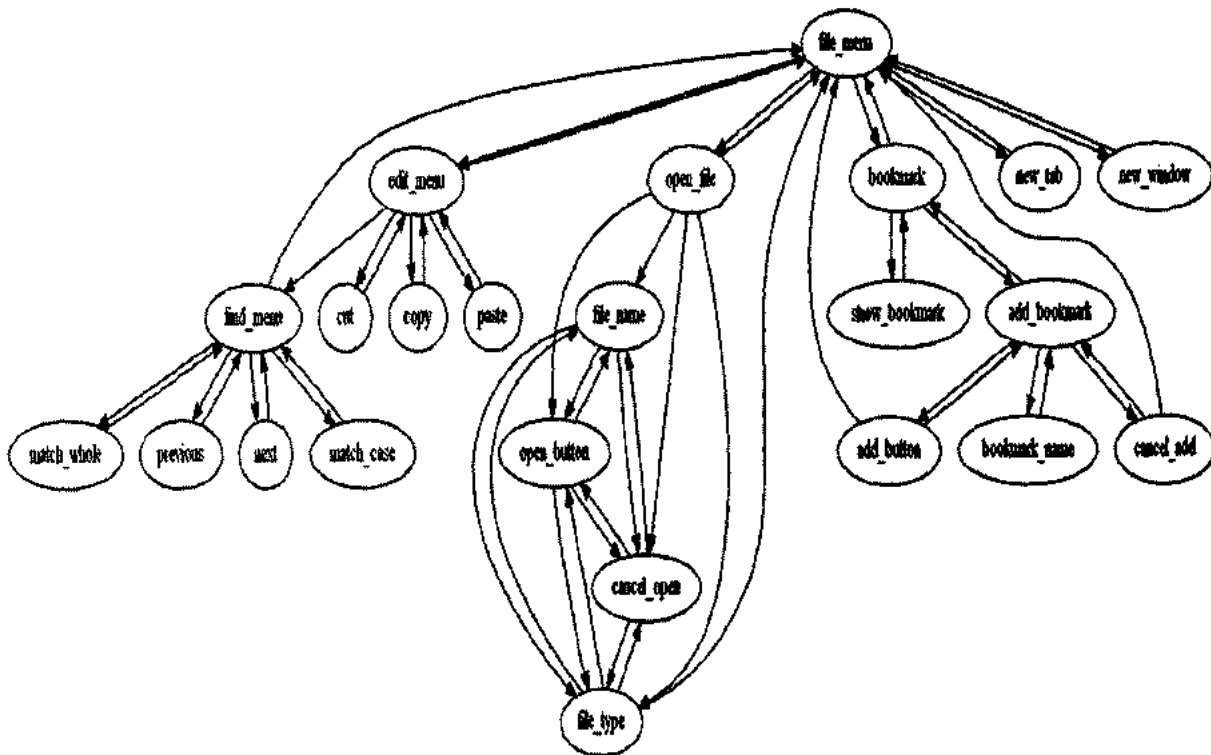


Fig 7.4 EFG of Internet Explorer (IE)

7.4 EXPERIMENTAL RESULTS OF EVENT COVERAGE AND EVENT-INTERACTION COVERAGE

Experimental Results of Event Path Coverage:

Path 1: file_menu -> open_file -> file_type -> open_button -> cancel_open -> file_name -> cancel_add -> add_bookmark -> bookmark

Path 2: file_menu -> edit_menu -> find_menu -> match_whole -> find_menu -> match_case -> find_menu -> next -> find_menu -> previous -> find_menu

Path 3: file_menu -> open_file -> file_type -> open_button -> cancel_open -> file_name -> cancel_add -> add_bookmark -> add_button -> add_bookmark -> bookmark_name -> add_bookmark -> bookmark -> show_bookmark -> bookmark

Path 4: file_menu -> edit_menu -> copy -> edit_menu -> paste -> edit_menu -> cut -> edit_menu -> paste -> edit_menu -> copy -> edit_menu -> cut -> edit_menu

Path 5: file_menu -> new_window

Path 6: file_menu -> new_tab

No of paths generation	Event coverage in Proposed approach
Path1	9
Path2	6
Path3	3
Path4	3
Path5	1
Path6	1
Total number of events	23

Table 7.3 Event Coverage of Internet Explorer (IE)

Experimental Results of Edge Path coverage:

Edge Path 1: (file_menu-open_file) -> (open_file-file_menu)

Edge Path 2: (file_menu-edit_menu) -> (edit_menu-find_menu) -> (find_menu-file_menu)

Edge Path 3: (file_menu-bookmark) -> (bookmark-file_menu)

Edge Path 4: (file_menu-open_file) -> (open_file-file_type) -> (file_type-file_menu)

Edge Path 5: (file_menu-edit_menu) -> (edit_menu-file_menu)

Edge Path 6: (file_menu-new_window) -> (new_window-file_menu)

Edge Path 7: (file_menu-new_tab) -> (new_tab-file_menu)

Edge Path 8: (file_menu-bookmark) -> (bookmark-add_bookmark) -> (add_bookmark-bookmark) -> (bookmark-file_menu)

Edge Path 9: (file_menu-open_file) -> (open_file-cancel_open) -> (cancel_open-file_type) -> (file_type-open_button) -> (open_button-file_type) -> (file_type-file_name) -> (file_name-file_type) -> (file_type-file_menu)

Edge Path 10: (file_menu-edit_menu) -> (edit_menu-find_menu) -> (find_menu-file_menu)

Edge Path 11: (file_menu-open_file) -> (open_file-open_button) -> (open_button-cancel_open) -> (cancel_open-open_button) -> (open_button-file_name) -> (file_name-open_button) -> (open_button-file_type) -> (file_type-open_button) -> (open_button-cancel_open) -> (cancel_open-file_name) -> (file_name-cancel_add) -> (cancel_add-file_menu)

Edge Path 12: (file_menu-edit_menu) -> (edit_menu-file_menu)

Edge Path 13: (file_menu-bookmark) -> (bookmark-add_bookmark) -> (add_bookmark-cancel_add) -> (cancel_add-add_bookmark) -> (add_bookmark-add_button) -> (add_button-file_menu)

Edge Path 14: (file_menu-new_window) -> (new_window-file_menu)

Edge Path 15: (file_menu-new_tab) -> (new_tab-file_menu)

Edge Path 16: (file_menu-open_file) -> (open_file-file_name) -> (file_name-file_type) -> (file_type-file_name) -> (file_name-open_button) -> (open_button-file_name) -> (file_name-cancel_add) -> (cancel_add-file_menu)

Edge Path 17: (file_menu-edit_menu) -> (edit_menu-paste) -> (paste-edit_menu) -> (edit_menu-copy) -> (copy-edit_menu) -> (edit_menu-cut) -> (cut-edit_menu) -> (edit_menu-find_menu) -> (find_menu-match_whole) -> (match_whole-find_menu) -> (find_menu-match_case) -> (match_case-find_menu) -> (find_menu-next) -> (next-find_menu) -> (find_menu-previous) -> (previous-find_menu) -> (find_menu-file_menu)

Edge Path 18: (file_menu-bookmark) -> (bookmark-show_bookmark) -> (show_bookmark-bookmark) -> (bookmark-file_menu)

Edge Path 19: (file_menu-open_file) -> (open_file-file_menu)

Edge Path 20: (file_menu-edit_menu) -> (edit_menu-file_menu)

Edge Path 21: (file_menu-bookmark) -> (bookmark-add_bookmark) -> (add_bookmark-bookmark_name) -> (bookmark_name-add_bookmark) -> (add_bookmark-bookmark) -> (bookmark-file_menu)

Edge Path 22: (file_menu-new_window) -> (new_window-file_menu)

Edge Path 23: (file_menu-new_tab) -> (new_tab-file_menu)

Edge Path 24: (file_menu-open_file) -> (open_file-file_type) -> (file_type-file_menu)

Edge Path 25: (file_menu-edit_menu) -> (edit_menu-find_menu) -> (find_menu-file_menu)

Edge Path 26: (file_menu-bookmark) -> (bookmark-add_bookmark) -> (add_bookmark-cancel_add) -> (cancel_add-add_bookmark) -> (add_bookmark-add_button) -> (add_button-add_bookmark) -> (add_bookmark-bookmark) -> (bookmark-show_bookmark) -> (show_bookmark-bookmark) -> (bookmark-file_menu)

No of path generation	Unvisited edge coverage in Proposed approach
Path1	2
Path2	3
Path3	2
Path4	2
Path5	1
Path6	2
Path7	2
Path8	2
Path9	6
Path10	8
Path11	2
Path12	2
Path13	2
Path14	14
Path15	2
Path16	2
Path17	1
Total number of edges covered	55

Table 7.4 Event-interaction Coverage of Internet Explorer (IE)

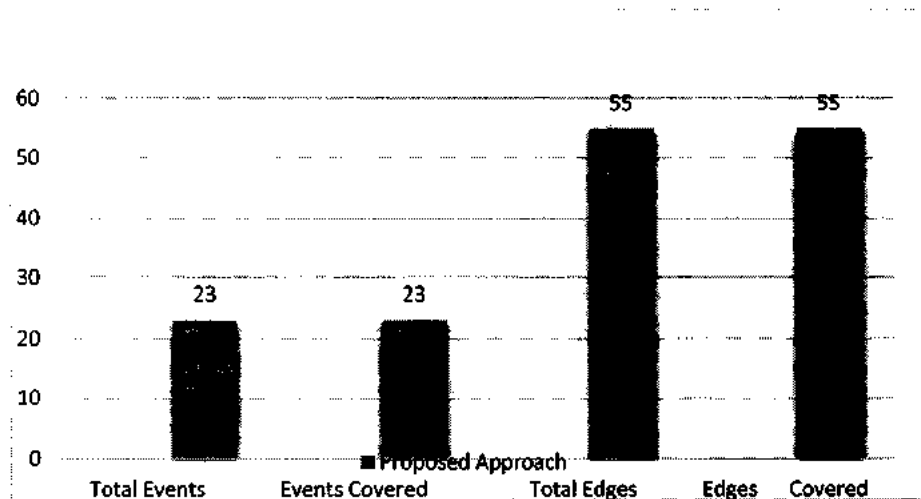


Fig. 7.5 Coverage criterion of proposed approach

From the resulted test paths of events it is concluded that all events in the EFG are visited at least once. As shown in the generated paths there is no event unvisited left behind so from these test paths it is concluded that event coverage criterion is satisfied.

As discussed earlier that event coverage criterion is not enough to detect the faults in software system so we have to choose stronger coverage criterion than the simple event coverage criterion. For this purpose event-interaction (event-pair) coverage criterion is used. This criterion ensures that all the events and the edges of the event flow graph are traversed. If all edges or event pair relation are visited this confirms that every event and every edge is covered so these are less chance of error left. Since all the events and edges between events are covered it means that functionality of the software system is tested fully and the system will become fault tolerant.

7.5 CASE STUDY 3

The 3rd case study is about the GUI of Windows media player. Media player is a Graphical user interface which provides ease to user to play the audio and video files. The user can listen songs, video clips, audio clips etc. The user can also add the favorite clips into favorite list and many more. Figure below is the event flow graph of windows media player in which there are 30 events and 67 follow relation events.

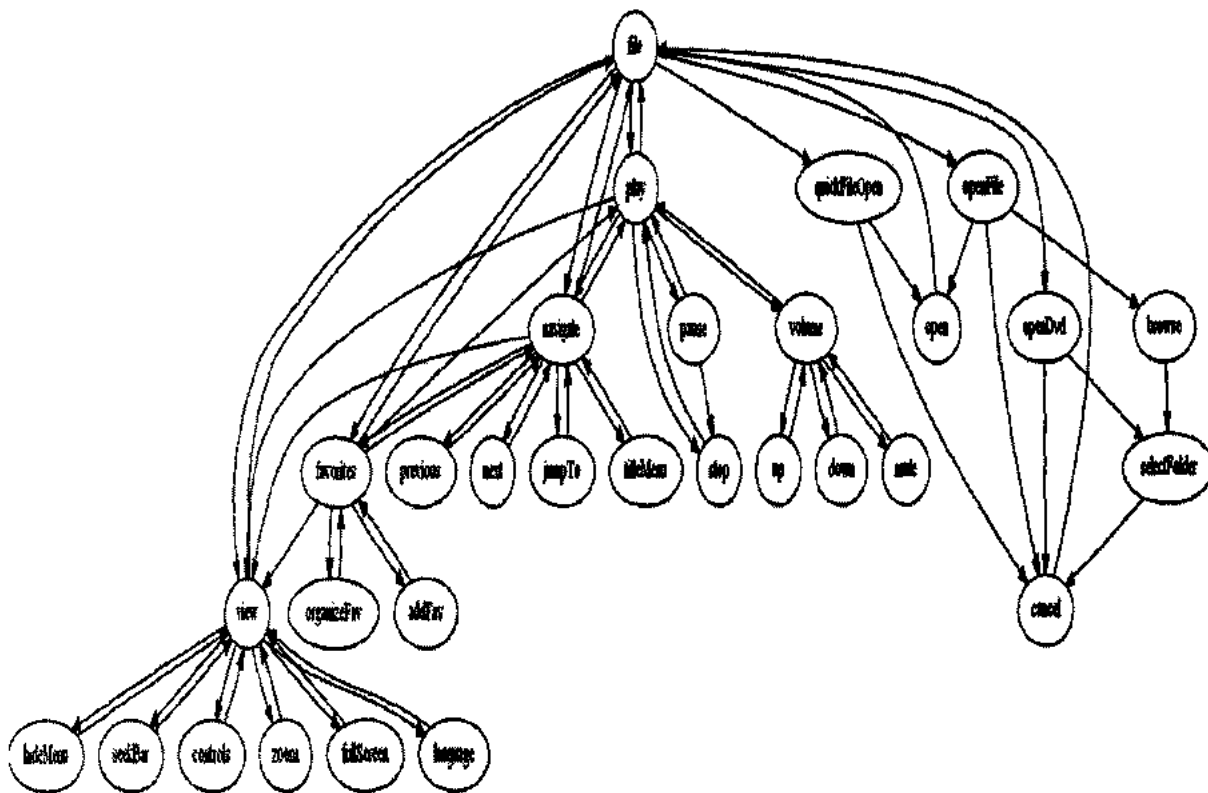


Fig 7.6 Event Flow Graph (EFG) Example of windows Media player

7.5.1 Generated Test paths

The output of the generated tests paths in the form of events and edges are given below which cover all events and completely explore all user interaction among events.

Event Path: file -> navigate -> view

Event Path: file -> play -> volume -> play -> navigate -> favorites -> view

Event Path: file -> open_file -> browse -> select_folder -> cancel

Event Path: file -> favorites -> navigate -> view -> language -> view -> full_screen -> view -> zoom -> view -> controls -> view -> seek_bar -> view -> hide_menu -> view

Event Path: file -> open_dvd -> select_folder -> cancel

Event Path: file -> quick_file_open -> open

Event Path: file -> play -> pause -> stop -> play -> volume -> mute -> volume -> down -> volume -> up -> volume -> play -> navigate -> favorites -> add_fav -> favorites -> organize_fav -> favorites -> navigate -> title_menu -> navigate -> jump_to -> navigate -> next -> navigate -> previous -> navigate -> title_menu -> navigate -> jump_to -> navigate -> next -> navigate -> previous -> navigate -> play -> pause -> stop -> play -> pause -> stop -> play

7.6 CASE STUDY 4

Another case study which enhance our work strength and validation is a Gtalk which is a web based application. Gtalk is a social messenger in which different user interact with each other, share files and other stuff. The user can also add new contacts in your contact list. Figure below is the event flow graph of Gtalk in which there are 24 events and 45 follow relation events.

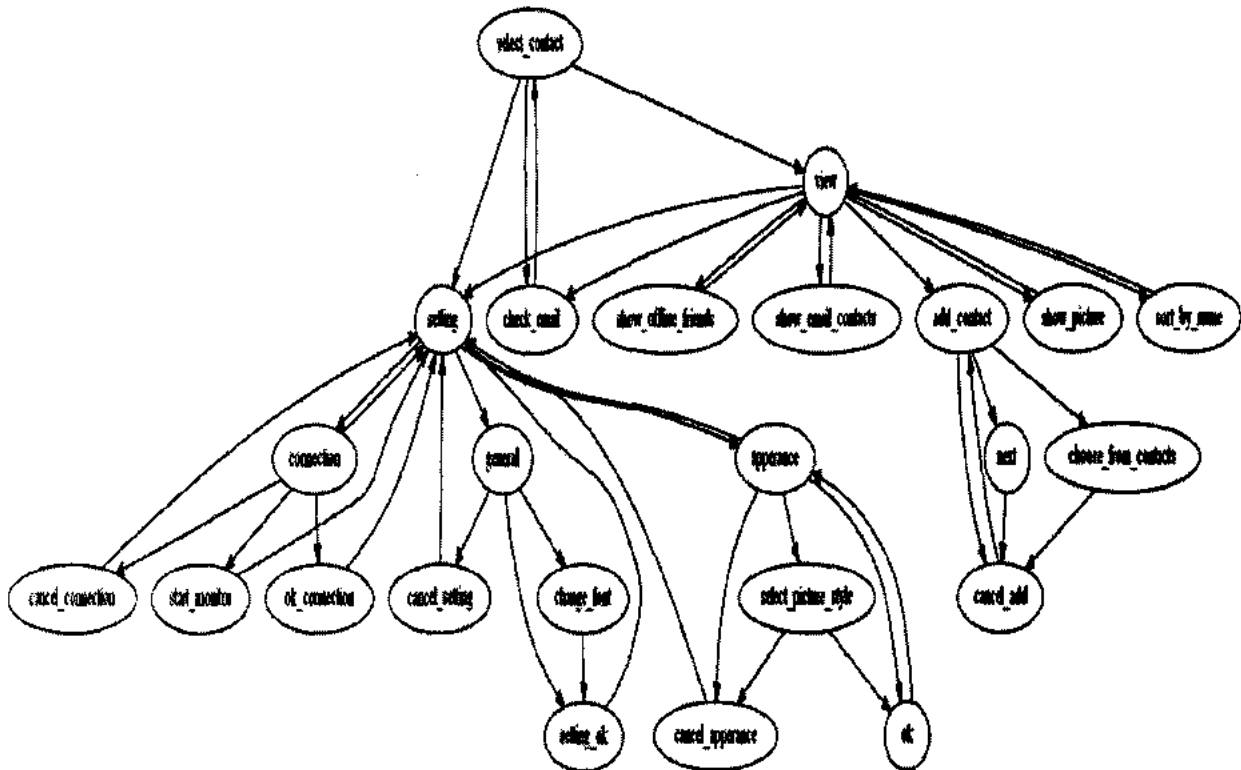


Fig 7.7 Event Flow Graph (EFG) Example of Gtalk

7.6.1 Generated Test paths

Event Path: select_contact -> view -> setting -> apperance -> select_picture_style -> cancel_apperance -> setting -> connection -> ok_connection -> setting -> general -> cancel_setting -> setting -> apperance -> ok -> apperance -> select_picture_style -> cancel_apperance -> setting -> connection -> start_monitor -> setting

Event Path: select_contact -> view -> add_contact -> next -> view -> show_picture -> view -> check_email

Event Path: select_contact -> view -> show_email_contacts -> view -> show_offline_friends -> view -> sort_by_name -> view -> add_contact -> cancel_add -> add_contact -> choose_from_contacts -> cancel_add -> add_contact -> next -> view -> show_picture -> view -> check_email

Event Path: select_contact -> view -> show_email_contacts -> view -> show_offline_friends -> view -> sort_by_name -> view -> add_contact -> choose_from_contacts -> cancel_add -> add_contact -> next -> cancel_add -> add_contact -> next -> view -> setting -> general -> change_font -> setting_ok -> setting -> connection -> cancel_connection -> setting -> apperance -> ok -> apperance -> select_picture_style -> cancel_apperance -> setting -> general -> cancel_setting -> setting -> connection -> ok_connection -> setting -> general -> change_font -> setting_ok -> setting -> connection -> start_monitor -> setting -> apperance -> select_picture_style -> ok -> apperance -> select_picture_style -> cancel_apperance -> setting

Edge Path:

Edge Path: (select_contact-view) -> (view-setting) -> (setting-apperance) -> (apperance-setting) -> (setting-connection) -> (connection-setting) -> (setting-general) -> (general-setting_ok) -> (setting_ok-setting) -> (setting-select_contact)

Edge Path: (select_contact-setting) -> (setting-apperance) -> (apperance-select_picture_style) -> (select_picture_style-cancel_apperance) -> (cancel_apperance-setting) -> (setting-connection) -> (connection-setting) -> (setting-general) -> (general-change_font) -> (change_font-setting_ok) -> (setting_ok-setting) -> (setting-select_contact)

Edge Path: (select_contact-view) -> (view-add_contact) -> (add_contact-next) -> (next-view) -> (view-select_contact)

Edge Path: (select_contact-view) -> (view-setting) -> (setting-apperance) -> (apperance-setting) -> (setting-connection) -> (connection-ok_connection) -> (ok_connection-setting) -> (setting-general) -> (general-cancel_setting) -> (cancel_setting-setting) -> (setting-select_contact)

Edge Path: (select_contact-setting) -> (setting-apperance) -> (apperance-cancel_apperance) -> (cancel_apperance-setting) -> (setting-connection) -> (connection-start_monitor) ->

(start_monitor-setting) -> (setting-general) -> (general-setting_ok) -> (setting_ok-setting) -> (setting-select_contact)

Edge Path: (select_contact-view) -> (view-show_picture) -> (show_picture-view) -> (view-show_email_contacts) -> (show_email_contacts-view) -> (view-show_offline_friends) -> (show_offline_friends-view) -> (view-sort_by_name) -> (sort_by_name-view) -> (view-check_email) -> (check_email-select_contact)

Edge Path: (select_contact-check_email) -> (check_email-select_contact)

Edge Path: (select_contact-view) -> (view-add_contact) -> (add_contact-cancel_add) -> (cancel_add-add_contact) -> (add_contact-choose_from_contacts) -> (choose_from_contacts-cancel_add) -> (cancel_add-add_contact) -> (add_contact-next) -> (next-view) -> (view-select_contact)

Edge Path: (select_contact-setting) -> (setting-apperance) -> (apperance-ok) -> (ok-apperance) -> (apperance-setting) -> (setting-connection) -> (connection-cancel_connection) -> (cancel_connection-setting) -> (setting-general) -> (general-change_font) -> (change_font-setting_ok) -> (setting_ok-setting) -> (setting-select_contact)

Edge Path: (select_contact-view) -> (view-setting) -> (setting-apperance) -> (apperance-select_picture_style) -> (select_picture_style-ok) -> (ok-apperance) -> (apperance-setting) -> (setting-connection) -> (connection-setting) -> (setting-general) -> (general-cancel_setting) -> (cancel_setting-setting) -> (setting-select_contact)

Edge Path: (select_contact-view) -> (view-add_contact) -> (add_contact-cancel_add) -> (cancel_add-add_contact) -> (add_contact-choose_from_contacts) -> (choose_from_contacts-cancel_add) -> (cancel_add-add_contact) -> (add_contact-next) -> (next-view) -> (view-select_contact)

Edge Path: (select_contact-setting) -> (setting-apperance) -> (apperance-cancel_apperance) -> (cancel_apperance-setting) -> (setting-connection) -> (connection-setting) -> (setting-apperance) -> (apperance-ok) -> (ok-apperance) -> (apperance-select_picture_style) -> (select_picture_style-cancel_apperance) -> (cancel_apperance-setting) -> (setting-connection) -> (connection-ok_connection) -> (ok_connection-setting) -> (setting-general) -> (general-setting_ok) -> (setting_ok-setting) -> (setting-select_contact)

Edge Path: (select_contact-view) -> (view-setting) -> (setting-apperance) -> (apperance-setting) -> (setting-connection) -> (connection-start_monitor) -> (start_monitor-setting) -> (setting-general) -> (general-change_font) -> (change_font-setting_ok) -> (setting_ok-setting) -> (setting-select_contact)

Edge Path: (select_contact-check_email) -> (check_email-select_contact)

Edge Path: (select_contact-setting) -> (setting-apperance) -> (apperance-setting) -> (setting-connection) -> (connection-cancel_connection) -> (cancel_connection-setting) -> (setting-general) -> (general-cancel_setting) -> (cancel_setting-setting) -> (setting-select_contact)

Edge Path: (select_contact-view) -> (view-show_picture) -> (show_picture-view) -> (view-show_email_contacts) -> (show_email_contacts-view) -> (view-show_offline_friends) -> (show_offline_friends-view) -> (view-sort_by_name) -> (sort_by_name-view) -> (view-check_email) -> (check_email-select_contact)

Edge Path: (select_contact-view) -> (view-add_contact) -> (add_contact-next) -> (next-view) -> (view-select_contact)

Edge Path: (select_contact-setting) -> (setting-appearance) -> (appearance-select_picture_style) -> (select_picture_style-ok) -> (ok-appearance) -> (appearance-setting) -> (setting-connection) -> (connection-setting) -> (setting-general) -> (general-setting_ok) -> (setting_ok-setting) -> (setting-select_contact)

Edge Path: (select_contact-view) -> (view-setting) -> (setting-appearance) -> (appearance-cancel_appearance) -> (cancel_appearance-setting) -> (setting-connection) -> (connection-setting) -> (setting-general) -> (general-change_font) -> (change_font-setting_ok) -> (setting_ok-setting) -> (setting-select_contact)

Edge Path: (select_contact-view) -> (view-setting) -> (setting-appearance) -> (appearance-ok) -> (ok-appearance) -> (appearance-setting) -> (setting-connection) -> (connection-setting) -> (setting-general) -> (general-cancel_setting) -> (cancel_setting-setting) -> (setting-select_contact)

Edge Path: (select_contact-setting) -> (setting-appearance) -> (appearance-select_picture_style) -> (select_picture_style-cancel_appearance) -> (cancel_appearance-setting) -> (setting-connection) -> (connection-ok_connection) -> (ok_connection-setting) -> (setting-appearance) -> (appearance-setting) -> (setting-connection) -> (connection-start_monitor) -> (start_monitor-setting) -> (setting-general) -> (general-setting_ok) -> (setting_ok-setting) -> (setting-select_contact)

Edge Path: (select_contact-view) -> (view-add_contact) -> (add_contact-cancel_add) -> (cancel_add-add_contact) -> (add_contact-choose_from_contacts) -> (choose_from_contacts-cancel_add) -> (cancel_add-add_contact) -> (add_contact-next) -> (next-cancel_add) -> (cancel_add-add_contact) -> (add_contact-cancel_add) -> (cancel_add-add_contact) -> (add_contact-choose_from_contacts) -> (choose_from_contacts-cancel_add) -> (cancel_add-add_contact) -> (add_contact-next) -> (next-view) -> (view-select_contact)

Chapter 8

Results and Discussion

8 RESULTS AND DISCUSSION

In this chapter we will describe and calculate the appropriate ability of the proposed approach. This chapter includes the comparison of the existing and the proposed approach then presents the results of proposed approach.

In this research we have covered the event coverage and event-interaction coverage criteria which calculate all paths having events and their edge coverage. This coverage ensures that all events and their follow-relation event are covered from all edge views.

In the previous chapter we discussed the parameters and how we performed the experiment in detail. We also showed the results of selected examples after performing experimentation. Now in this chapter we will compare our results of proposed approaches with Huang et al. results.

8.1 ANALYSIS OF EXPERIMENTAL RESULTS

When ACO is performed on the Event flow Graph (EFG) all events that are selected are constructed based on calculating the probability of every event. Then we find out the results that are given below.

We also implemented the Huang et al. work and found out the results, and then we compared our work with the work of Huang et al. work. Results show that event coverage gives the same result as Huang et al. [33] proposed, but the event-interaction coverage criteria gives better results than Huang et al. results. It means that our proposed method are better than Huang et al. method.

For easy illustration, EFG example in Fig. 7.1 is used for experiment result. The EFG of Notepad contain 17 events along with the 61 follows relationship events as shown in the Fig. 7.1

We can see the results of Fig.7.1 that all 17 events and 61 follow relations are visited. Huang et al. only covered 35 out of 61 follow relations. Existing research used simple graph traversal algorithm to generate test cases. Therefore some paths of EFG are not executable due to different event context. Like if we use Depth First Search (DFS) algorithm the selected path to PasteMenuItem is FileMenuItem->EditMenuItem->PasteMenuItem->CopyMenuItem. The selected path is infeasible because PasteMenuItem is not enabled. PasteMenuItem can only be enabled when CopyMenuItem is executed before it. To overcome the infeasible problem the depending paths are identified earlier and describe their depending relation.

	Total Events	Events Covered	Total Edges	Edges Covered
Huang et al.	17	17	61	35
Proposed Approach	17	17	61	61

Table 8.1 Comparison of Experimental Results

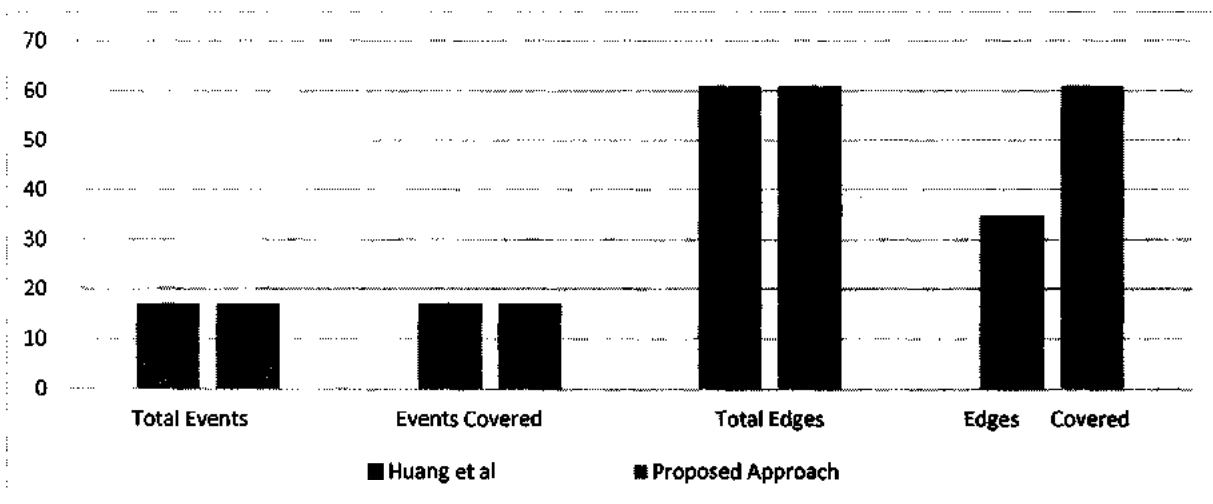


Fig.8.1 Comparison of Event Coverage and Event-interaction Coverage

As shown in the table 8.1 that the event coverage fulfill the criteria of covering all the events at least once. In the Fig. 7.1 there are total 17 events in the EFG and the resulted output also covers the 17 events this means the event coverage is satisfied. Existing approach generated one long path to cover all events but in our approach we have generated multiple paths which has also covered all the events. In the existing methodology there are total 17 events and 61 event-pair relation. To achieve the full coverage that make the software fault free we have to cover all these 17 events and 61 event-pair relation (follow relation events). In our proposed approach these criteria are enhanced and instead of covering 35 out of 61 follow relation, 61 follow relation events are covered. From the above table it is concluded that our proposed approach is better than the existing approach.

Case study 2	Total Events	Events Covered	Total Edges	Edges Covered
Proposed Approach	23	23	55	55

Table 8.2 Coverage criterion for internet explorer

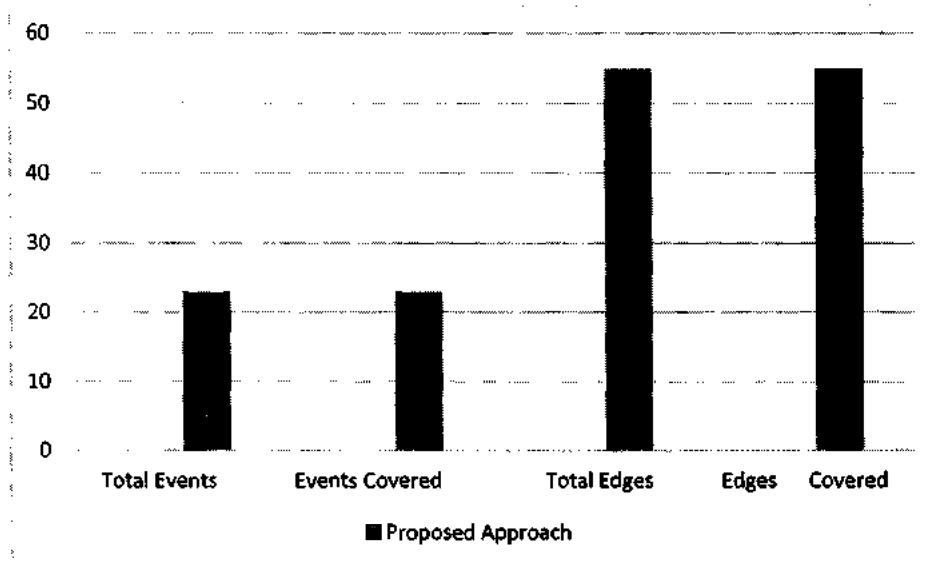


Fig 8.3 Coverage criterion of proposed approach for Case study 2

From the above table 8.2 it is concluded that the proposed approach covers all events along with the event-interaction coverage. This coverage criterion is stronger than the simple event coverage criterion because it covers the edges and events both.

Simple event coverage might miss some edges or follow-relation of the events. That's why for adequate testing we need stronger coverage criterion which provides full events and full edge coverage. From the experimental result it is shown that the existing methodology only covers events of the Event Flow Graph (EFG) leaving some follow relation unvisited. But the proposed approach covers all events of the EFG and also provides full coverage to traverse the edges of the EFG. Hence proved from the experimental result that the coverage criterion in the proposed approach is better than the existing approach.

There are many GUI path coverage criteria that are used for GUI testing. Among all of them Event coverage criterion covers all the events which shows the GUI objects and components. For stronger GUI testing Coverage criterion event-interaction

coverage criterion is used that covers all the edges which shows the follow-relation events (Child events). In event coverage all events are covered but might be some edges are left uncovered. These uncovered edges might produce faults in software system. To cover these edges which represent the dependencies between different edges, event-interaction coverage criterion is used. It is proved from the case studies that existing approach used only event coverage so there is a change of error. To overcome this missing or uncovered edges, event- interaction coverage is used. The test path covers all feasible events and all their feasible edges.

Chapter 9

Conclusion and Future work

9 CONCLUSION AND FUTURE WORK

In conclusion we summarize our main influence of thesis work and also suggests some future work.

9.1 CONCLUSION

GUI testing is a vital phase in development of software which confirms the software quality and reliability. Automated GUI testing makes software testing more efficient and less error prone. In our work we are implementing feasible test paths generation. A lot of work has done on test data generation along with the coverage criterion and generation of feasible test paths. One of the authors worked on coverage criterion for effective testing but does not perform the full coverage for event flow graph (EFG).

So, our main contribution is to generate the test path having full coverage and eliminating the infeasible paths using Ant Colony Optimization (ACO) algorithm. In proposed approach event-interaction coverage criterion is used to provide full coverage which covers events and their adjacent edges (child events). We perform the experimentation to validate our methodology. We compare our results with other work Huang et al. Our methodology are effective and efficient than huang et al approach. Existing approach provide only events coverage while missing some edges uncovered. Our hypothesis accepted that accuracy is improved due to event-interaction coverage criterion. Feasible path generation and event-interaction coverage using ACO are automated.

9.2 FUTURE WORK

In future work we consider other GUI coverage criterion for effective test path generation on complex and huge case study. Our main focus is generation of feasible test paths. We will also consider the infeasible paths repairing in future work. We have selected initial event but we are not considering the initial event randomly. So, it will be also our future

work that we will select initial node randomly on the basis of event availability. This means event is not depending upon the other event when selection of event is random.

Generation of all coverage criterion and all feasible paths from program is automated but event flow graph representation in XML form is not automated which will consider as future work for automation.

References

REFERENCES

- [1] Beizer Boris. "*Software Testing Techniques*", ISBN-13: 978-1850328803 New York. 1990.
- [2] Hamlet, Dick. "*Foundations of software testing: dependability theory.*" ACM SIGSOFT Software Engineering Notes. Vol. 19. No. 5. ACM, 1994.
- [3] Zhu, Hong, Patrick AV Hall, and John HR May. "*Software unit test coverage and adequacy.*" *Acm computing surveys (csur)* vol 29.4 pp. 366-427, 1997.
- [4] Tracey, Nigel, et al. "*An automated framework for structural test-data generation.*" Automated Software Engineering, 1998. Proceedings. 13th IEEE International Conference on. IEEE, 1998.
- [5] Whittaker, James A. "What is software testing? And why is it so hard?." *Software*, IEEE vol 17.1 pp. 70-79, 2000.
- [6] Memon, Atif M. "A comprehensive framework for testing graphical user interfaces". Diss. University of Pittsburgh, 2001.
- [7] Belli, Fevzi. "*Finite state testing and analysis of graphical user interfaces.*" *Software Reliability Engineering*, 2001. ISSRE 2001. Proceedings. 12th International Symposium on. IEEE, 2001.
- [8] Memon, Atif M., Mary Lou Soffa, and Martha E. Pollack. "*Coverage criteria for GUI testing.*" ACM SIGSOFT Software Engineering Notes vol. 26.5 pp. 256-267, 2001.

-
- [9] Memon, Atif M., Martha E. Pollack, and Mary Lou Soffa. *"Hierarchical GUI test case generation using automated planning."* Software Engineering, IEEE Transactions on 27.2 pp. 144-155, 2001.
- [10] White, Lee, Husain Almezen, and Nasser Alzeidi. *"User-based testing of GUI sequences and their interactions."* Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on. IEEE, 2001.
- [11] Memon, Atif M. *"GUI testing: Pitfalls and process."* Computer vol. 35.8 pp. 87-88, 2001
- [12] Li, Huaizhong, and Chiou Peng Lam. *"Software Test Data Generation using Ant Colony Optimization."* International Conference on Computational Intelligence. 2004.
- [13] Memon, Atif M. Adithya Nagarajan, and Qing Xie. *"Automating regression testing for evolving GUI software."* Journal of Software Maintenance and Evolution: Research and Practice vol. 17.1 pp. 27-64 , 2005.
- [14] Dorigo, Marco, Mauro Birattari, and Thomas Stutzle. *"Ant colony optimization."* Computational Intelligence Magazine, IEEE vol. 1.4 pp. 28-39, 2006.
- [15] Li, Ping, Toan Huynh, Marek Reformat, and James Miller. *"A practical approach to testing GUI systems."* Empirical Software Engineering vol. 12.4 pp. 331-357, 2007.
- [16] Memon, Atif M. *"An event-flow model of GUI based applications for testing."* Software Testing, Verification and Reliability vol.17, No.3, pp.137-157, 2007.

-
- [17] Naik K &Tripathy P. "*Software Testing and Quality Assurance Theory and Practice*", ISBN-13: 978-0471789116 , John Wiley & Sons, 2008
- [18] Yaseen, Saad Ghaleb, and Nada MA AL-Slamy. "*Ant colony optimization.*" IJCSNS Vol.8 No.6 pp.351, 2008.
- [19] Lu, Yongzhong, Danping Yan, Songlin Nie, and Chun Wang. "*Development of an improved GUI automation test system based on event-flow graph.*" Computer Science and Software Engineering, 2008 International Conference on. Vol. 2. IEEE, pp. 712–715, 2008.
- [20] Srivastava, Praveen Ranjan, K. M. Baby, and G. Raghurama. "*An approach of optimal path generation using ant colony optimization.*" TENCON 2009-2009 IEEE Region 10 Conference. IEEE, 2009.
- [21] Li, Kewen, Zilu Zhang, and Wenying Liu. "*Automatic test data generation based on ant colony optimization.*" 2009 Fifth International Conference on Natural Computation. Vol. 6. 2009.
- [22] Jin, Hu, Shuo Wang, Nian-Wei Chen, and Zhen Ye. "*Finite State Machine for Automatic GUI Testing.*" Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on. IEEE, 2009.
- [23] Yuan, Xun, and Atif M. Memon. "*Iterative execution-feedback model-directed GUI testing.*" Information and Software Technology Vol. 52 No.5 pp.559-575, 2010.

-
- [24] Rauf, Abdul, Sajid Anwar, M. Arfan Jaffer, and Arshad Ali Shahid. "Automated GUI test coverage analysis using GA." In Information Technology: New Generations (ITNG), 2010 Seventh International Conference on, pp. 1057-1062. IEEE, 2010.
- [25] Miao, Yuan, and Xuebing Yang. "An FSM based GUI test automation model." Control Automation Robotics & Vision (ICARCV), 2010 11th International Conference on. IEEE, 2010.
- [26] Yuan, Xun, and Atif M. Memon. "Generating event sequence-based test cases using GUI runtime state feedback." Software Engineering, IEEE Transactions on Vol.36 No.1 pp. 81-95, 2010
- [27] Gulia, Preeti, and R. S. Chillar. "A new approach to generate and optimize test cases for UML state diagram using genetic algorithm" ACM SIGSOFT Software Engineering Notes Vol.37 No.3 pp. 1-5, 2010.
- [28] Myers, Glenford J., Corey Sandler, and Tom Badgett. "The art of software testing". ISBN-13: 978-1118031964 John Wiley & Sons, 2011.
- [29] Yuan, Xun, Myra B. Cohen, and Atif M. Memon. "GUI interaction testing: Incorporating event context." Software Engineering, IEEE Transactions on vol. 37.4 pp. 559-574, 2011.
- [30] Rauf, Abdul, Arfan Jaffar, and Arshad Ali Shahid. "Fully automated GUI testing and coverage analysis using Genetic Algorithms." International Journal of Innovative Computing, Information and Control (IJICIC) Vol. 7, 2011.

-
- [31] Retna and Emi "*Study paper on the test case generation for GUI based testing.*" International Journal of Software Engineering & Applications Vol.3 No.1, 2012.
- [32] Suri, Bharti, and Shweta Singhal. "*Literature survey of Ant Colony Optimization in software testing.*" Software Engineering (CONSEG), 2012 CSI Sixth International Conference on. IEEE, 2012.
- [33] Huang Y and L. Lu. "Apply ant colony to event-flow model for graphical user interface test case generation." IET software Vol.6 No.1 pp. 50-60, 2012.
- [34] Khamis, Abdelaziz M, Moheb R. Girgis, and Ahmed S. Ghiduk. "*Automatic software test data generation for spanning sets coverage using genetic algorithms.*" Computing and Informatics Vol.26 No.4. pp. 383-401, 2012.