Data Flow Testing of State Machine Using Ant Colony Algorithm (ACO)

Submitted by:

Fozia Mehboob

FBAS/MSSE/F09

Supervised by:

Mr Atif Aftab Ahmed Jilani

Co-Supervised by:

Mr Imran Saeed

Department of Computer Science & Software Engineering

FACULTY OF BASIC & APPLIED SCIENCES

International Islamic University H-10 Islamabad

MSE
005.7
FOD

1. Data in computer system

2. Database management

Department of Computer Science & Software Engineering

International Islamic University Islamabad

Date: ----------------

Final Approval

It is certified that we have read this research thesis report and have fully evaluated the research undertaken by **Fozia Mehboob, Registration No. 279/FBAS/MSSE/F09**.This research thesis fully meets the requirement of Department of Computer Science & Software Engineering , and hence, the International Islamic University, Islamabad for the degree of **Master of science in Software Engineering (MSSE).**

Committee:

External Examiner:

Prof. Dr. Sikandar Hayat Khiyal

APCOMS, Khadim Hussain Road,

Lalkurti, Rawalpindi

[illegible]

Internal Examiner:

Dr. Zunera Jalil

Assistant Professor, DCS & SE, FBAS, IIUI

Faculty of Basic & Applied Sciences

International Islamic University Islamabad/IIUI

[illegible]

Supervisor	*Co-Supervisor*
Mr Atif Aftab Ahmed Jilani	Mr Imran Saeed
Assistant Professor	Assistant Professor
Faculty of Computer Science	Department of CS and SE
FAST/National University of Computer & Engineering Sciences	IIUI/International Islamic University
[illegible]	[illegible]
------------------------------	------------------------------

Dedication...

I would like to dedicate my research work

To my

FATHER "Malik Ghulam Mehboob"

Whose Sincere Love and prayers were a source of

Strength for me and made me to do this research work

Successfully.

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 Fulfilment of the Requirements for the Award of the

Degree of Master of Science in Software Engineering (MSSE).

DECLARATION

I hereby declare that this Thesis **"Data Flow Testing of State Machine using Ant Colony Algorithm (ACO)"** neither as a whole nor as a part thereof has been copied out from any source. It is further declared that I have written this thesis entirely on the basis of my personal effort, made under the proficient guidance of my thesis supervisor Mr Atif Aftab Ahmed Jilani. If any part of this research thesis proved to be copied or found to be a research of some other individual, I shall standby the consequences.

Fozia Mehboob

279-FBAS/MSSE/F09

Acknowledgement

In the name of ALLAH, the most Compassionate and the most merciful whose guidance made me so able to conduct and compiled this research work. It is God bestowed pride that I was able to accomplish it successfully. Thanks to **Almighty ALLAH for His** guidance and Vision so granted to me.

Every project has an objective attached to it which generates new and creative ideas. I am lucky that intelligence of my supervisor Mr Atif AftabAhmed Jilani, served as guiding star. My thesis would have not been possible in the absence of such intellectual guidance. His worthy advices, honest supervision and affable attitude are worth revealing. I therefore attribute my Master degree to him because of his support and effort and have no words to show gratitude to him.

I would like to articulate my gratitude to Sir Imran Saeed for his sincere guidance and kind cooperation. I would also like to acknowledge Dr. Abdul Rauf for his genuine support, and motivation throughout the project. I would also like to thanks Dr Uzair for his worthy advices and sincere comments. Despite his tight schedules, he gave his valuable time to document of thesis. I thank also my committee members for their commendable comments and criticism.

I would like to pay my deepest gratitude to my parents for their constant support, encouragement for the achievement of my work. Their prayers are always in overcoming difficult tasks. It is due to their care and affection for that I am at this stage today. Most especially, my father is responsible for my curiosity and always encourage me in my tough times to pull me up.

I am thankful to my caring brother Qasim Ali for constantly helping me during hard times and offering me his valuable advice. My Grandmother deserves special appreciation for providing me moral support and prayers. Last but not least, I would like to gratitude my sweet friends especially Sobia Noreen for her love, care , moral support and boosting me to overcome my morale if and when I was depressed due to strainful work.

Every research work can't be perfect and final. I accept accountability for all deficiencies if any in this dissertation. I shall be appreciative for worthy suggestions and would welcome optimistic criticism.

Fozia Mehboob

279-MSSE/FBAS/F09

Project In Brief

Project Title:	Data Flow Testing using Ant Colony Optimization
Undertaken By:	Fozia Mehboob
	279-FBAS/MSSE/F09
Supervised By:	Mr Atif Aftab Ahmed Jilani
Start Date:	June 01, 2011
Completion Date:	September 06, 2012
Tools & Technologies	Microsoft Visual Studio 2008
	IBM Rational Software Architect
	Enterprise Architect UML Modelling
Documentation Tools	Microsoft Office Word 2007
Operating System:	Microsoft Windows 7, Home Premium
System Used:	HP Probook 4530s Notebook PC

Abstract

Automatic data flow testing refers to analysis of flow of data within models by using data flow analysis rules. To ensure correct data flow within states we have to consider these data values. The data flow analysis forms a source of testing data flow by considering define and uses of the variables. Empirical studies have shown that existing state- based approaches are not efficient in detecting state based faults. State-based testing examines state changes and its behaviour without focusing on the internal details, thus data faults remain uncovered. However, to completely analyze data flow within models and fulfil the coverage criteria is not an obvious task, especially in state based systems. The complex nature of state based models further aggravates the situation, making the data flow testing problem complicated and time consuming.

It has been observed that many state based approaches don't provide complete definition-use path complete coverage and also are ineffective in terms of detection of data flow faults. Our work begins by considering all these observations to view automatic data flow analysis problem and solve with heuristic technique. In this research work, a novel approach is presented for automatic data flow testing of UML state machine models and automatically generates test cases. We view data flow testing problem as an optimization problem and select optimal number of feasible test cases to provide complete def-use paths coverage. Extensible Mark-up Language (XML) of state machine models is used to given as input. Data flow information is extracted from XML. After the system extract data flow information, variables are categorized as defined or used. Def-Use pairs and def-use paths are identified. Automated feasible test cases are generated from UML models. Minimal test cases provide the complete coverage of definition-use paths. A best possible solution is investigated by make use of the heuristic search technique Ant Colony Optimization Algorithm. We implemented this approach in a tool that is named as data flow generator (DFG).

As a proof concept, applicability of this approach is checked by applying it to different UML state machine models representing the dynamic system behaviour. Experiments are performed to validate this approach which indicates that maximum complete def-use path coverage can be obtained. However, the only prerequisite of this approach is to have a XML of given input models. Furthermore, we can use a small size and non-exhaustive UML state machine models. Moreover, the proposed approach is effective enough in automatic detection of data flow errors. This is to a certain extent not possible in existing state based system approaches in which undetection of these data flow errors results in incomplete coverage of def-use paths.

Conclusively, by automatic detection of test paths from state based systems, our approach provides complete data flow analysis. Moreover, mutation analysis is performed to analyze our proposed approach effectiveness. In this way, our proposed approach makes data flow analysis process of models painless by automatically data flow errors and improve or enhance the effectiveness of fault detection of state based systems.

Table of Contents

Abstract ix

List of Figures xviii

List of Tables xx

Acronyms and Abbreviations xxi

Chapter 1. Introduction **1**

1.1 Introduction 2

1.2 Problem Statement 2

1.3 Motivation 3

1.4 Research Questions 4

1.5 Proposed Solution 4

1.6 Thesis Contribution 5

1.6.1 Contribution 6

1.7 Dissertation Outline 6

Chapter 2. Background **8**

2.1 Introduction 9

2.2 State Based Testing 9

2.2.1 UML Models 9

2.3 Data Flow Testing 10

2.3.1 Coverage Criteria 10

2.4 Automated Test Case Generation 11

2.5 Swarm Intelligence 11

2.5.1 ACO 12

Chapter 3 Related Work **14**

3.1 Introduction 15

3.2 Data Flow Testing Approaches 15

3.2.1 Code Based Data Flow Testing Approaches 15

3.2.2 Model Based Data Flow Testing Approaches 16

3.2.3 Data Flow Testing of UML State Machine Using Metaheuristic Approaches 16

3.2.3.1 Code Based Data Flow Testing Using Metaheuristic Approaches 16

3.2.3.2 Model Based Data Flow Testing Using Metaheuristic Approaches 18

3.3 Data Flow Testing Tools 18

3.3.1 Code Based Data Flow Testing Tool 18

3.4 Data Flow Testing Using Swarm Intelligence 20

3.5 Analysis 21

3.5.1 Limitations in Research Work 22

Chapter 4. Problem Definition **23**

4.1 Introduction 24

4.2 Issues of State Based Testing 24

4.3 Concern of Data Flow Testing Tool 24

4.4 Gap 25

Chapter 5. Proposed Approach 27

5.1 Introduction 28

5.2 Preliminaries 28

5.2.1 Input Source Model 28

5.2.2 UML State Machine Models 28

5.2.3 XML 29

5.3 Approach Overview 29

5.4 Test Data Selection 33

5.4.1 UML State Machine Models 33

5.5 ACO Adaptation for Automated Data Flow Testing 35

5.5.1 Representation of Solution 36

5.5.2 Evaluation of Optimal Solution 37

5.5.3 Deriving an Optimal Solution 41

5.5.4 Parameter Tuning 43

5.6 Automated Data Flow Testing 43

5.6.1 Prepare the Input Source Mode 44

5.6.2 Create Control Flow Graph 44

5.6.3 Identify Data Flow Information 45

5.6.4 Automatically Generate Feasible Test Cases 45

5.6.5 Data Flow Information against each Path........45

5.6.6 Optimum Set of Test Cases Using Optimal Solution........45

Chapter 6. Tool Implementation........46

6.1 Introduction........47

6.2 Tool Architecture........47

6.2.1 XML Parser........47

6.2.2 Search Engine........49

6.2.3 Automated Test Case Generator Engine........49

6.3 Tool Implementation........50

6.3.1 Path........51

6.3.2 ACO........52

6.3.3 Graph........52

6.3.4 Variables........54

6.3.5 Editor........55

6.3.6 Nodes........55

6.3.7 Errors........56

6.3.8 XML Parser........56

6.4 Tool Process Flow........57

6.4.1 Input Source Model........58

6.4.2 Parameter Setting........59

6.4.3 Run ACO 60

6.4.3.1 Create Adjacency Matrix 62

6.4.3.2 Generate Feasible Test Sequences 63

6.4.3.3 Generate Infeasible Test Cases 64

6.4.3.4 Generate Optimal Test Cases 65

Chapter 7. Case Study 71

7.1 Introduction 72

7.2 Elevator Control System 72

7.2.1 Scope of Elevator Control System 72

7.2.2 Functional Requirement 73

7.2.3 Details of Case Study 74

7.2.3.1 Infeasible Paths 82

7.2.3.2 Mutation Testing 82

Chapter 8. Evaluation 84

8.1 Introduction 85

8.2 Elevator Control System 85

8.2.1 Experimental Settings of Tool 85

8.2.2 Results and Discussion 86

8.3 Validation of our Proposed Approach 98

8.3.1 Experimental Settings 98

8.3.2 Results and Discussion of Validation of Proposed Approach 99

8.4 Comparisons 100

8.4.1 Data Flow Testing Approaches 100

8.5 Assessment 101

8.5.1 Benefits 101

8.5.2 Limitations 102

Chapter 9. Conclusion & Future Work **103**

9.1 Introduction 104

9.2 Conclusion 104

9.3 Future Work 106

9.3.1Improve Optimal Set of Test Cases 106

9.3.2 Application to Large Scale Models 106

9.3.3 Reduce the Test Case Generation Time 106

9.3.4 Enhance Data Flow Testing Tool 106

9.3.5 Applicability of our Proposed Approach 106

9.3.6 Applicability of Approach to other Models 107

References 108

Appendix A. XML Format of all Input Models 112

Appendix B. Screen Shots all Input UML State Machine Models 122

Appendix C. User Manual 132

Appendix D. Generated Test Cases & Data Flow Information 142

List of Figures

Chapter 1

Figure 1....................................Format of XML 5

Figure 1.2.................................... Dissertation Outline 7

Chapter 5

Figure 5.1....................................Format of XML 29

Figure 5.2................................Data Flow Testing System 30

Figure 5.3........................Data Flow Testing Approach Overview... 31

Figure 5.4...........................Process Flow of Proposed Approach... 32

Figure 5.5.....................State Machine Diagram of Telephone System 34

Figure 5.6....................................Adjacency Matrix 35

Figure 5.7..........................Steps of Automated Data Flow Testing Process.................................... 44

Chapter 6

Figure 6.1...Tool Architecture 48

Figure 6.2..Class Diagram.. 50

Figure 6.3..Main Window of Tool... 57

Figure 6.4..................................XML File of Input Model of ECS 58

Figure 6.5...Main GUI.. 59

Figure 6.6.................................Main GUI (Select Run ACO Option) 60

Figure 6.7....................................Adjacency Matrix of ECS Model 62

Figure 6.8...........................Feasible Test Cases of Input Model ECS 63

Figure 6.9...............................Optimal Trial 1 of Input Model ECS... 64

Figure 6.10 .... Optimal Trial 2 of Input Model ECS 65

Figure 6.11 ....Optimal Trial 3 of Input Model ECS 66

Figure 6.12 ....Optimal Trial 4 of Input Model ECS 67

Figure 6.13 ....Optimal Trial 5 of Input Model ECS 68

Figure 6.14 ....Optimal Trial 6 of Input Model ECS 69

Figure 6.15 ....Optimal Trial 7 of Input Model ECS 70

Chapter 7

Figure 7.1.......................Adjacency Matrix of Elevator State Machine Model.................................. 75

Figure 7.2.........................State Machine Diagram of Elevator Control System................................ 76

Chapter 8

Figure 8.1.......................................XML Format of ECS 86

Figure 8.2.........................Bar Chart of Def-Use Paths Vs Mutation Score 87

Figure 8.3.........................Line Chart of Def-Use Paths Vs Mutation Score..................................... 87

Figure 8.4................................Mutated Telephone System Model.. 90

Figure 8.5.............Automatically Generated Test Cases of Telephone System Model 97

List of Tables

Chapter 2

Table 2.1 .... Steps of ACO 12

Chapter 5

Table 5.1 .... Def-use pairs and Def-clear paths 37

Chapter 7

Table 7.1 .... Def-use Pairs of the Variables used within State machine Diagram 81

Table 7.2 .... Showing Mutation Score of all Models 83

Chapter 8

Table 8.1 .... DU-Paths Coverage & Mutation Scores of all Models 87

Table 8.2 .... Type and Number of Error Seeded 91

Table 8.3 .... Mutation Score & No. of Test Cases Needed 91

Table 8.4 .... Fault-Wise Coverage Provided by Existing Techniques 92

Table 8.5 .... General Coverage Provided by Existing Code-Based Techniques 93

Table 8.6 .... General Coverage Provided by Existing Model-Based Techniques 94

Table 8.7 .... Total Def-use pairs & Def-use paths in Paper [10] 95

Table 8.8 .... Number of Input Models and DU-pairs 99

Acronyms and Abbreviations

XML	Extensible Markup Language
UML	Unified Modelling Language
SM	State Machine
CFG	Control Flow Graph
DFT	Data Flow Testing
DU	Definition-Use
ACO	Ant Colony Optimization
PSO	Particle Swarm Optimization
GA	Genetic Algorithm
BFS	Breadth First Search
ECS	Elevator Control System
TS	Telephone System
HMS	Hospital Management System
LMS	Library Management System
POS	Purchase Order System
DM	Display Manager
SES	Student Enrollment System
CC	Cruise Control
ATM	Automated Teller Machine

Chapter 1

INTRODUCTION

1.1 Introduction

In software engineering, development of software is highly competitive task to provide software product of high quality within specific constraints. Software testing is vital part in development of software. Exhaustive testing is impossible; to achieve the testing effectiveness testing process should be automated. In software testing, test data generation is a critical task in formation of test cases that fulfill the coverage criteria.

Recently researchers focus on to automatically generate test cases to reduce the cost having high quality and unproblematic software testing [10]. To measure adequacy of testing, many researchers use different well known coverage criteria defining the stopping condition of testing process, e.g. data and control flow criteria. In Software, generation of test cases mainly focuses the information regarding control flow for the production of optimum test cases. Testing of data-flow is significant because it supplements control flow information leading to more efficient as well as targeted test cases. Data flow analysis analyzes, works out the relationship and association among data objects. Existing data flow testing approaches and tools aren't efficient in terms of 1) detection of data flow faults, [9, 31] 2) incomplete coverage of data flow information, [9, 31] 3) redundancy in test cases [9, 11, 31]. Currently data flow testing of UML state machine is performed fulfilling the required coverage criteria result in generation of optimal number of non redundant test cases.

1.2 Problem Statement

Recently researchers focus on to automatically generate minimal test cases that are based on data flow criteria. Consequently, in literature there are many approaches that perform data flow analysis and testing of state based systems along with control-flow information to thoroughly analyze the software system. By the analysis of these existing state based approaches, it is observed that all the approaches are not efficient in detection of data flow faults and fulfilling the coverage criteria.

Automated data flow testing in state based systems has many limitations that confine the results. Some of these limitations are: 1) some of the approaches are ineffective in terms of fault detection [9], 2) incomplete coverage of all def-use paths [9, 31], 3) moreover, generation of large number of redundant test cases result in time consumption, 4) some of the approaches perform data flow testing of state machine models don't generate test cases to cover def-use paths [10]. All these difficulties result in incomplete data flow analysis of complex state machine models. Due to all these reasons, data flow analysis and non redundant generation of optimal number of test cases becomes a demanding activity. Data flow analysis of variables and their relationship within states are considered only. No such approach exists that perform data flow analysis 1) using ACO, 2) fulfilling data flow oriented coverage criteria within minimal number of paths, 3) and efficient in fault detection.

Motivation

Taking account of only information regarding control flow of models in model based testing is not adequate to make sure, whether flow of data is properly through model [10]. Most of existing state-based approaches focused on the control flow structure and don't examine state changes and its behaviour. No existing state-based testing approaches performing data flow testing provide efficient detection of data faults. However, currently the approaches perform data flow testing using metaheuristic approach don't provide all def-use paths coverage; generate a number of test cases result in redundant and infeasible test cases because of data flow errors are not completely detected.

From all these observations and keeping in view all these limitations, our work starts to perform data-flow testing of UML state machine,. The motivation of this research is to put forward an approach that improves existing state based coverage criteria. And provide a mechanism to select the Optimum set of test sequences among alternative while ensuring all definition-use paths complete coverage. Manual removal of redundant and infeasible test cases is a time consuming task; result in incomplete coverage of all def-use paths. In this way, automatic generation of non-redundant test cases is done to fulfill the definition-use paths coverage in addition to efficient fault detection to evade the difficulties associated with manual detection.

1.3 Research Questions

This research work plan to deals with the subsequent three research questions:

1. How effective is Data Flow testing using Ant Colony Algorithm in enhancing fault detection capability?
2. How Data Flow testing using Ant Colony Algorithm provides complete definition-use paths coverage?
3. How Data Flow testing using Ant Colony Algorithm is effective in reducing the search space?

1.4 Proposed Solution

Observance of all the troubles and complication linked with existing state based data flow testing approaches/techniques; we propose an approach that makes the data flow testing of state based system simple, and uncomplicated. The main idea of proposed approach is to generate minimal test paths in fulfilling data flow coverage criteria and provides maximum data flow error detection.

We propose to use state based models to perform data flow testing. In our approach, XML is given as input to tool that is commonly used representation in model based automated testing tools. XML is generated using UML modeling tool enterprise architect. Both the data-flow and information about control flow are extracted from XML of given input state machine model. The format of XML is given in fig 1.1. Control flow information is collected is shown in adjacency matrix to get the feasible connection between states. Data flow analyses of variables within states are performed to know the relationship between variables and how they affect flow of execution in models.

In state based testing, generation of test cases become difficult due to large number of test cases ineffective in fault detection. Most of the state based approaches focus on control flow information doesn't examine state changes and its behavior, Moreover, approaches providing coverage of data flow information is incomplete. In this way, we provide an approach automatically generate optimal number of test cases that not only ensure all def-use paths as well

3. Redundant test cases are minimized.
4. Reduces the search space.

The benefits obtained due to developing an approach having the characteristics that are given below.

1. Ease the automated generation of test cases from state machine models.
2. Provides the complete coverage of all definition-use paths with minimal test cases which is rather impossible in existing state based approaches.
3. Efficient detection of data flow errors.
4. Generation of non-redundant paths.
5. Tool is efficient in complete detection of all seeded faults within models.

1.6.1 Contribution

None of the existing model based approaches provides complete analysis of data flow faults. Our approach use metaheuristic technique to perform data flow analysis that not only provides effectiveness in detection of data flow faults and with optimal number of test sequences, coverage of all def-use paths is provided. Due to use of heuristic technique, all seeded faults are also effectively detected by tool. Tool provides general (state coverage) as well as fault-wise completes coverage of models.

1.6 Dissertation Outline

Figure 1.2 demonstrates the association of this dissertation: Chapter 2 set up the background to understand the dissertation by providing the knowledge regarding data flow testing. Related work in the field of data flow testing of UML state machine is described in chapter 3. Chapter 4 highlights and demonstrates the issues and limitations of existing state based approaches and also research gap is highlighted. Our proposed approach that automatically perform data-flow testing of UML state machine is described in chapter 5. Tool implementation of proposed approach is described in chapter 6. To validate our proposed approach, a case study is presented in chapter 7.

2.5.1 Ant Colony Optimization (ACO)

Ant colony optimization (ACO), a famous research area in paradigm of swarm intelligence. ACO algorithm was proposed by Dorigo et al in 1990's derive from behavior of real ants [19]. This algorithm solves the travelling salesman problem. ACO algorithm is based on the idea of sensing pheromone traces on paths and bases their decision on this value, to search the shortest/optimal paths within a graph [18].

Ant Colony Algorithm has been by the behavior real ant's colonies, observing them when they are searching for food source. An ant lays a substance known as pheromone on a path that is traversed by ants in searching of food source. This pheromone information helps other ants moving randomly by marking the path and also returns to their original source by using this information. Path that is followed by more ants have more pheromone value because new ants also lays pheromone on that specific path update its pheromone value. As pheromone lay by ants evaporate at a constant rate that helps avoidance of convergence to local optima [20].

```
Initialization
– Initialize the ant's position
Iteration
For each Ant do
– On the Basis of probability choose the next state to move into;
– Repeat until all ants completed a solution;
– Update the pheromone values for each path that are traversed by ants;
Update the Graph;
End;
If global solution is not better than local solution
keep best local solution as global solution
End;
```

Table 2.1 Steps of ACO

The more ants pursue the same path, the more pheromones value is on that path. As a result, there will be larger probability for the ants to go for on the same path. ACO simulates the mutual aid process of real ant colony. Each ant leaves pheromones on solution it gets by independently searching for solution. Paths having higher pheromones values have larger probability to be chosen by ants [1]. Each ant makes use of the graph to search the optimal solution within graph. Each ant has its own memory having start state and termination condition. Ant's decision is probabilistic decision based on available pheromone trails [17]. ACO is a strong heuristic approach that involves positive feedback leads to the near best solutions in minimal time. ACO technique in selection gives better results at higher test case values with minimal amount of time taken by the algorithm [19].

ACO has been used in solving the various combinatorial optimization problems such as knapsack problem, travelling salesman problem, distributed networks, data mining, telecommunication networks, vehicle routing, test data and test suite selection [19].

Chapter 3

RELATED WORK

3.1 Introduction

Search-Based generation of test cases is a promising methodology for the automatic generations of test data of high quality. Using evolutionary algorithms to automatically generate test cases is widely used by researchers. To select and generate quality test data from meta-heuristic algorithms are used. Test cases are selected within the search space that satisfies a certain testing criteria.

These chapters of dissertation discuss the existing work in this field. Our work can be associated with two fields of software engineering, Data flow testing of state machine models and application of swarm intelligence in data flow testing. Section 3.2 illustrated the approach, techniques and tools related to data flow testing of UML state machines. Section 3.3 presents the tools for testing the data flow within Models. Data flow testing using Swarm intelligence is explained in section 3.4. Lastly section 3.5 explained the conclusion of chapter by analyzing existing approaches and tools.

3.2 Data flow Testing Approaches

Initially researchers focused on data flow testing of code based programs. After some time, they performed data flow testing of models and automatically generate test cases. We categorize these existing literature based on their focus on models and on code coverage.

3.2.1 Code-Based Data Flow Testing Approaches

Frankl et al [48], performed data flow testing of programs and also focused on feasible criteria for data flow testing. This Coverage criterion exercises only those deflnition-use pairs that are executable. This is a code based approach and our focus is on models.

Weyuker's et al [50] focused on number of criteria to select a test data. All-definition, all uses, all edges, all c-use, all p-use and all definition-use paths coverage criteria are evaluated on program. Analysis indicates that data flow faults remained undetected in all edges and all-defs criteria. Coverage of all c-use doesn't include all-edges and some of the paths aren't covered by these criteria that are covered by all definition-use paths coverage criteria. This is a code based approach and our focus is on models.

Girgis et al [37] presents a data-flow testing technique and focused on def-use paths. This technique was not effective as the value of fitness function for all test cases was same; test cases covering same number of def-use pairs and that don't cover any def-use path. And use nodes aren't covered by any test case. This is a code based approach and our focus is on models.

3.2.2 Model-Based Data Flow Testing Approaches

Briand et al [9] performed data flow testing and focused on transition tree coverage criteria. A data flow criterion was not used to create test cases but they analyze already created test suites to choose best among them. Their approach provides incomplete def-use pair coverage and there are infeasible paths due to incompatible sequence of transition. They manually remove infeasible paths. Our proposed approach is extension of this approach because it provides complete def-use paths coverage and also minimizes redundant generation of paths. Secondly, this approach doesn't use any heuristic approach for analysis of flow of data within model.

Waheed et al [10] performs data flow analysis and use action semantics to analyze data flow of variables and dependencies. They give state machine as an input and feasible path matrix is created to collect the information regarding control flow. Approach includes the data flow information by finding the def-use relationship between the variables that are within states of given input model. Our approach is different with this approach in the sense that it automatically generates test cases along with detection of def-use pairs within model. Secondly mutation testing is not performed in [10] that our proposed approach do so.

3.2.3 Data flow Testing of UML State Machine using Meta-heuristic Approaches

Different code-based and model-based approaches use metaheuristic techniques to perform data flow testing of UML State Machine are given in subsection 3.2.3.1 and 3.2.3.2.

3.2.3.1 Code-Based Data flow Testing using Meta-heuristic Approaches

Khor et al [14] presented an automatic test case generation approach known as genet while focusing on branch coverage criteria. Genet used genetic algorithm for test case generation and formal concept analysis was used to organize the relationship between tests. They evaluate the effectiveness of their approach with random test generation technique called randy. ATGs provide better coverage with less number of test cases. This is a code based approach and our focus is on model based data flow testing using metaheuristic approach.

Andreou et al [6] proposed a technique for data flow coverage criteria that combine existing testing framework with data flow graph. To automatically generate test cases, genetic algorithm was used and focused on All Definition-use path coverage criteria. Valuable information was extracted by their Basic program analyzer, which creates the control flow and data flow graph. Experiment was conducted on java programs of various size and complexity. Comparison was conducted by comparing technique with existing data flow generator techniques, showed better experimental results as compared to existing techniques. This is a code based approach and our focus is on model based data flow testing using metaheuristic approach.

Girgis et al [28] proposed technique that generates test cases focusing on all-uses coverage criteria. The technique uses genetic algorithm to evaluate the generated test cases. For defining a multiple objective fitness function, dominance relationship among nodes was considered. Technique is able to select specific test requirement, generating test cases to satisfy that requirement. To assess the efficiency of approach, they applied their approach on c++ programs. Limitation of their approach was applying it to large programs and size of population is small. Also time to search data depends on speed of machine that may vary and test cases generated in previous iteration are ignored. Random test case generation technique was used to compare their technique results. This is a code based approach and our focus is on model based data flow testing using metaheuristic approach.

Girgis et al [32] presented a technique to perform data flow testing of instrumented version of programs using genetic algorithm. All-uses criteria are used as coverage criteria. Roulette wheel method and random selection method was used for the selection of parents. Fitness function was calculated by counting the definition-use path covered within test cases and divided it to overall number of definition-use paths. Comparison of their technique was done with random testing technique. Test cases cover definition-use pairs are generated and also list down definition-use pairs that aren't covered. This is a code based approach and our focus is on model based data flow testing using metaheuristic approach.

Singla et al [3] presented automatic test generating technique for data flow testing while focusing on all-uses coverage criteria. Proposed algorithm use Particle Swarm Optimization to generate test cases and new fitness function was designed. Comparison of technique is done with Genetic Algorithm for all programs that are used in their experiment and with other data flow based testing techniques, results showed that PSO achieved higher coverage percentage than Genetic algorithm. This is a code based approach and our focus is on model based data flow testing using metaheuristic approach.

3.2.3.2 Model-Based Data flow Testing of UML State Machine using Meta-heuristic Approaches

Doungsa-ard et al [23] proposed a framework and implement a tool known as state diagram executor for test case generation from UML State machine. Transitions pairs covered by test cases are used as a coverage criteria and genetic algorithm was used for searching the high quality test data. Best solution was considered that covered maximum number of transitions but their approach produced better result when the system doesn't contain any final state. This approach only generates test cases but don't focus on analysis of data flow within model. Tool doesn't execute guard conditions if they aren't in mathematical expressions. And looping problems as well as infeasible transitions aren't handled by their approach. As this approach use heuristic technique only for generation of test cases but our proposed approach focus on analysis of data flow within models and also mutation testing is performed to analyze the effectiveness of our approach that above mentioned approach don't do so.

3.3 Data Flow Testing Tools

In literature, code based data flow testing tools were developed to detect the data flow errors within program that are given in subsection 3.3.1.

3.3.1 Code-Based Data Flow Testing Tools

Hou et al [8] analyze the flow of data and a BPEL tool was developed for testing of web services to detect automatically the data flow errors and generates paths which are based on certain data flow coverage criteria. The tool can automatically select the paths, without executing the process, result in incorrect data were detected. But tool don't generate test cases for selected paths automatically. This is a code based tool and our focus is on model based data flow testing tool.

Horgan et al [44] automatically designed test analysis tool to analyze the data flow in c programs. It creates new test cases that examine the code but test cases aren't generated automatically. It analyses the c source code and reads static data flow information for each c source code. To collect static data flow constructs, data flow graph is searched. Regression testing was also performed by this tool. This is a code based tool and our focus is on model based data flow testing tool.

Hamlet et al [45] constructed a prototype tool for analyzing the data flow information in arrays and record the executed paths during testing. A tool, Du-path analyzer was constructed to report the issues of data flow in arrays by considering element of each array as data flow object. This is a code based tool and our focus is on model based data flow testing tool.

Bluemke et al [24] performed code-based data flow testing of java programs by implementing tool known as data flow coverage tool is an eclipse plug-in. They focused on all defuse pairs coverage criteria and also def-use graph was created. All def-use pairs are found by testing the java code and also provide information about which def-use pairs are covered. Tool also detects data flow errors that aren't not be exposed by black box testing. This is a code based tool and our focus is on model based data flow testing tool.

Hou et al [8] developed a tool for data flow testing of web services that automatically detects data flow faults. Generation of test cases was done automatically focusing on data flow coverage criteria and generations of duplicate paths were avoided. Some of the features of BPEL tool were also supported by this tool. By using the data flow graph, it provides information of du-pairs, c-uses and p-uses as well as detects anomalies of data. This is a code based tool and our focus is on model based data flow testing tool.

3.4 Data Flow Testing using Swarm Intelligence

LI et al [33] used Ant Colony Optimization (ACO) algorithm for state-based software testing. They have developed a tool that convert the state machine diagram into directed graph and generates test cases to achieve all-states coverage criteria. A flattened state chart is used by the approach. Dynamic ant simulator which is a prototype tool was developed by using this approach to generate test cases automatically while avoiding redundant test cases due to use of ants. But this approach doesn't focus on data flow coverage criteria. Our Proposed approach also uses ACO but it is different with aforementioned approach in the sense that it uses all def-use path coverage criteria. Secondly, this approach use ACO only for generation of test cases but our proposed approach uses ACO to generate minimal paths and analyze def-use paths covered by these paths and also detects data flow errors that aforementioned approach doesn't do so.

Ranjan et al [11] generate test cases using Ant colony algorithm. They focused on criticality of states and average number of visits for optimizes test case generation. Comparison of technique was done with approach in [15]. Results indicate that their approach produced optimal number of test cases providing 100% critical states as well as edge coverage. But the work does not perform data flow testing. As mentioned above this approach use markov chain model to generate test cases but our proposed approach use state machine model for test case generation. Existing approach also don't consider data flow analysis that our proposed approach do so.

Lam et al [50] presented test case generation technique for state based testing using Ant colony optimization. Their work was same as in [11] but considered additional factors like criticality of states and average number of visits. Our approach differ with approach in a sense that above mentioned approach use ACO just for generation of test cases but our proposed approach uses ACO for data flow analysis of state based systems and analysis is performed using mutation testing.

3.5 Analysis

By the analysis of existing techniques, approaches and tools, we come to the subsequent conclusions:

1. As existing state-based techniques and approaches focus on control flow structure and examines state changes and its behavior without focusing on the internal details, thus data faults remain uncovered.
2. Main problems of state-based software testing approaches are that, infeasible test cases and redundant test paths are generated to fulfill the required coverage criteria [33].
3. Major focus of previous techniques performing data flow analyses are on code based techniques focusing on all-uses coverage criteria.
4. Approaches that are model based don't provide Def-Use Paths complete coverage.
   - Because they analyze the definition-use paths (du-paths) of test suites that are already created and consider that the test cases contains the feasible connection of states.
   - When they analyze the data flow in already created test cases, there were definition-use pairs (du-pairs) that were not covered and also undetected faults [9].
5. Some of the model based approaches performing data flow testing don't generate test cases considering the def-use pairs coverage [10].
6. Some of the model based techniques that use meta-heuristic technique, genetic algorithm for data flow testing focused in all-transitions pairs result in undetected infeasible paths.
7. Tools using swarm intelligence technique, ant colony algorithm generate test cases without focusing on data flow testing.

3.5.1 Limitations in Research Work

By examining existing literature, we come to conclusion that existing state based approaches are ineffective in detection of state based faults. Redundant test paths are generated to completely fulfill the coverage criteria. Mostly approaches which perform data flow analysis are code based. State-Based approaches that use heuristic techniques, generates only test cases while not focusing on data flow analysis. None of existing model based approaches use ACO for data analysis of state based system that our proposed approach does so. Due to this not only efficiency in fault detection is increased but also minimal test cases are generated to provide coverage.

Chapter 4

PROBLEM DEFINITION

4.1 Introduction

Search-Based generation of test cases is a methodology to automatically generate high quality test data. The use of evolutionary algorithms to generate test cases is widely used by researchers. For the selection and generation of quality test data, meta-heuristic techniques and algorithms are used. Test cases are selected within the search space that satisfies a certain coverage criteria.

This chapter of dissertation explains the limitation and issues of existing approaches for data flow testing using meta-heuristic techniques. Section 4.2 discusses the difficulties associated with automated testing of state based approaches. Issues of state based approaches are described in section 4.3.Section 4.4 elaborate the complexity of UML models. Existing research gap and problem statement is expressed in section 4.5.

4.2 Issues of State-Based Testing

There is extensive variation in cost in designing, executing test cases, and checking their effectiveness in terms of checking faults in existing state based approaches. Most of the criteria focused on control flow information of state diagram but some of the papers focused on data flow testing of state machine having limitations in their work [33]. Many coverage criteria are proposed for test cases generation from UML state machine, some of well known are including all transitions, all transitions pairs and all-paths etc. Empirical evaluation of these coverage criteria revealed that there is variation in cost in developing test cases. For example several trees are generated in all transition trees differing in fault detection rate, have low detection rate. While in all transition pair coverage criteria are extremely expensive but effective in detection of faults [9].

4.3 Concern of Data Flow Testing Tool

As there are infinite numbers of paths in a complex state machine diagram, it is impossible to select all of them for the determination of conformance of implementation to required behavior. To select an optimal solution, there is need coverage criteria to select optimal number of paths that satisfy certain conditions. Several testing methods are proposed in recent years, most of them focusing on control flow oriented coverage criteria without considering data flow information. Both type of information regarding data and control flow is necessary in model based testing to ensure correct flow of data through the model [10]. Generations of test cases from control and data flow selection criteria are complementary to each other [3, 32]. Both type of information must be used for selection of comprehensive tests. To automatically generate test cases based on certain coverage criteria has been a rapidly increasing interest for many researchers in recent years. Metaheuristic techniques are used for cost-effective generation of test cases automatically, using heuristic to seek an optimal solution for combinatorial problems [25].

4.4 The Gap

Existing gaps in research that forced this research are given below:

1. State based approaches focused on control flow structure without focusing on flow of data through the model.
2. State based approaches are not effective at detecting data flow faults.
3. Most of the existing approaches don't perform mutation testing to check the effectiveness of their approach.
4. Until now, there is no model based data flow testing tool providing complete def-use pair's coverage with optimal number of test sequence using meta-heuristic algorithm.
5. Existing approaches are ineffective in providing complete def-use paths coverage.
6. Currently some of the approaches which generate test sequences automatically using meta-heuristic techniques are code based focusing on data flow coverage.
7. Majority of existing code based approaches focusing both on data flow coverage, generate redundant test cases to provide coverage.
8. Most of the approaches that perform data flow testing of UML state machine don't encounter looping problem.
9. Some of the Model based approaches identify def-use pairs but don't generate test cases and perform mutation testing to analyze the effectiveness of their approach.
10. None of the Model Based approach use metaheuristic approach performs data flow testing of UML State Machine.

None of the existing techniques provides data flow coverage of model using Ant Colony Optimization algorithm focusing on both Optimal generation of test suites providing complete coverage of def-use pairs as well as automatic detection of data flow errors to ensure correct data flow through the model.

Chapter 5

PROPOSED APPROACH

5.1 Introduction

To improve the existing coverage criteria of state based approaches and to aim the research gap given in the previous chapter, an approach is proposed for automatic data flow testing. Our approach is considerably different from that already exist in current literature of state based testing. This approach makes use of swarm intelligence heuristic technique to automate the data flow testing process. These concepts have never been used in data flow testing context before.

This chapter is devoted to explain our proposed approach to perform automatic data flow testing. In the chapter, section 5.2 introduces basic concepts related to our approach. An overview of the approach is given in section 5.3. Section 5.4 provides details of test data. Using of ACO to identified problem of automatic data flow testing problem is presented in section 5.5. Section 5.6 explains the entire process of generating test sequences covering def-use pairs.

5.2 Preliminaries

We introduce some general concepts related to our approach to facilitate the further discussion.

5.2.1 Input Source Model

Input model is model of system that is given for producing the desired output. Input source model is UML state machine diagrams.

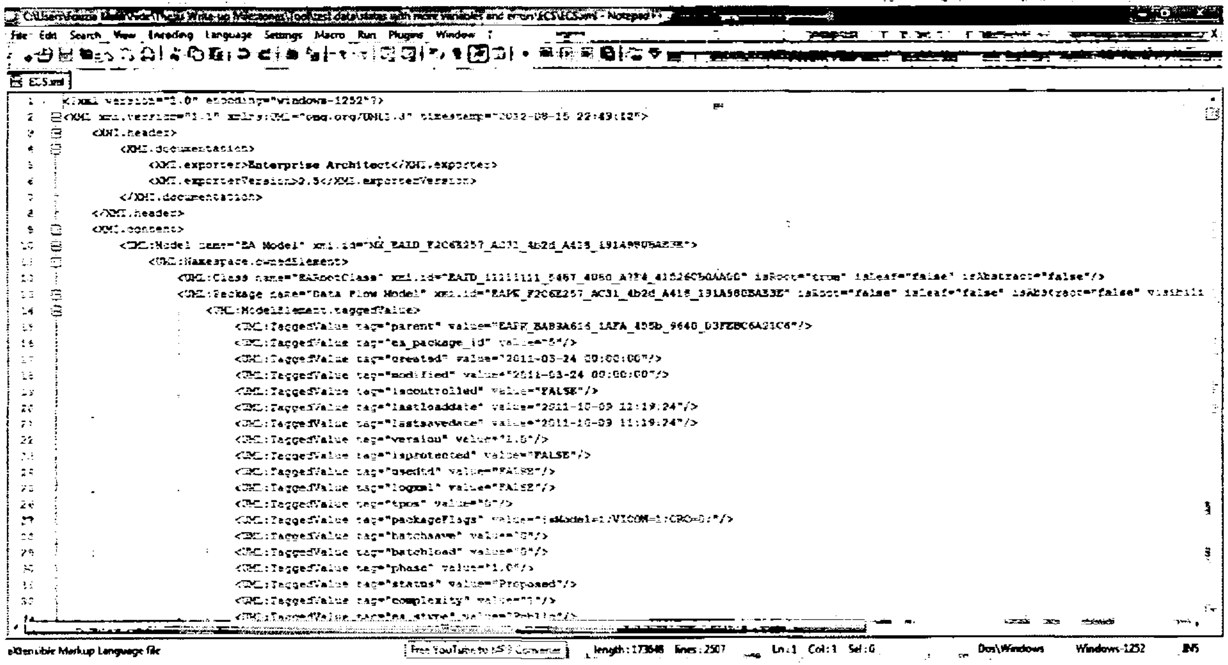
5.2.2 UML State Machine Models

UML state machine model consists of states, transitions showing connection between states and invariants. States may contain a number of variables and associations between them, etc. A complex state machine model consists of a number of states, paths and sub paths.

TH-10049

5.2.3 XML

In our approach, input of source model is given in the form of XML (Extensible Markup Language), which is commonly used in model based testing. XML is a markup language that encode document in a format that is human readable as well as machine readable by using set of rules defined in XML 1.0 Specification [28] that are produced by the W3C, and several others specifications etc. UML modeling tool enterprise architect is used to generate XML of state machine system models. XML contains the details information of overall states within input model, source and target transition of states, guard conditions, variables and relationship between them. This information is extracted by tool from XML to perform the data flow testing. The format of XML is given in Fig 5.1.


C:\Users\Fouzia ...\data\states with more variables and error\ECS\ECS.xml - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
ECS.xml
eXtensible Markup Language file
Free YouTube to MP3 Converter
length : 173648 lines : 2507
Ln : 1 Col : 1 Sel : 0
Dos\Windows
Windows-1252
INS

```
<?xml version="1.0" encoding="windows-1252"?>
<XMI xmi.version="1.1" xmlns:UML="omg.org/UML1.3" timestamp="2012-08-15 22:43:12">
    <XMI.header>
        <XMI.documentation>
            <XMI.exporter>Enterprise Architect</XMI.exporter>
            <XMI.exporterVersion>2.5</XMI.exporterVersion>
        </XMI.documentation>
    </XMI.header>
    <XMI.content>
        <UML:Model name="EA Model" xmi.id="MX_EAID_F2C6E257_AC31_4b2d_A418_191A980BA33E">
            <UML:Namespace.ownedElement>
                <UML:Class name="EARootClass" xmi.id="EAID_11111111_5487_4080_A7F4_41526CB0AA00" isRoot="true" isLeaf="false" isAbstract="false"/>
                <UML:Package name="Data Flow Model" xmi.id="EAPK_F2C6E257_AC31_4b2d_A418_191A980BA33E" isRoot="false" isLeaf="false" isAbstract="false" visibili
                    <UML:ModelElement.taggedValue>
                        <UML:TaggedValue tag="parent" value="EAPK_BAB3A616_1AFA_455b_9640_D3FEBC6A21C6"/>
                        <UML:TaggedValue tag="ea_package_id" value="5"/>
                        <UML:TaggedValue tag="created" value="2011-03-24 00:00:00"/>
                        <UML:TaggedValue tag="modified" value="2011-03-24 00:00:00"/>
                        <UML:TaggedValue tag="iscontrolled" value="FALSE"/>
                        <UML:TaggedValue tag="lastloaddate" value="2011-10-09 12:19:24"/>
                        <UML:TaggedValue tag="lastsavedate" value="2011-10-09 11:19:24"/>
                        <UML:TaggedValue tag="version" value="1.0"/>
                        <UML:TaggedValue tag="isprotected" value="FALSE"/>
                        <UML:TaggedValue tag="usedtd" value="FALSE"/>
                        <UML:TaggedValue tag="logxml" value="FALSE"/>
                        <UML:TaggedValue tag="tpos" value="0"/>
                        <UML:TaggedValue tag="packageFlags" value="isModel=1;VICON=1;CRC=0;"/>
                        <UML:TaggedValue tag="batchsave" value="0"/>
                        <UML:TaggedValue tag="batchload" value="0"/>
                        <UML:TaggedValue tag="phase" value="1.0"/>
                        <UML:TaggedValue tag="status" value="Proposed"/>
                        <UML:TaggedValue tag="complexity" value="1"/>
                        <UML:TaggedValue tag="ea_stype" value="Public"/>
```

Fig 5.1 Format of XML

5.3 Approach Overview

Main theme of our approach analyzes the data flow to perform data flow testing. Analysis of data is performed to find Definition-use pairs information between variables that exist within states. UML state machine models are used for automatic test case generation. States tags, source and target transitions ids, variable names and position within states are extracted from XML of

UML state machine. Adjacency matrix is created to create a control flow graph to automatically create control flow graph. Variables information is then used identify the relationship between variables, def-use pairs. Automated feasible set of test cases are generated covering data flow information.

Search based generation of test cases is example of search based software engineering. Many metaheuristic automated generations of test cases techniques have been developed that are proposed to deal with and generate quality of test data. In testing of data flow, selections of test paths are done on certain data flow criteria. Our approach use data-flow coverage criteria for automatic generation of test cases. Our data flow based system consists of two main components as shown in figure 5.2.

1. Coverage Criteria select the test data of quality which is evaluated against certain coverage provide by the test cases.
2. Search Engine is a software program that figure out information available in coverage criteria. We use heuristic search optimization technique as a search engine in our approach.

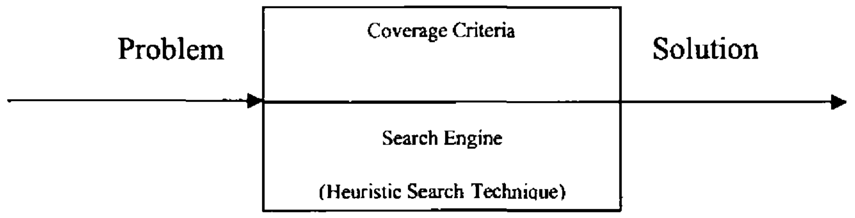


Figure 5.2 Data flow Testing System

In our approach, coverage criteria is all definition-use paths requires inclusion of test data that cause path traversal from each occurrence of variable definition to each of its use of variable. Our approach searches the all def-use pairs and paths within input model, removing redundant def-use paths. During search, along with generation of optimal number of paths providing maximum state as well as def-use pair coverage avoiding redundant test cases in each generation, data flow errors are also detected automatically. For that reason our approach differs from other data flow testing techniques.

By selecting minimal test paths which are based on maximum definition-use pair coverage of input model, the final solution consists of minimal test cases. Because input model consists of large number of transitions and states, have larger number of possible paths. As exhaustive testing is not possible to consider all the paths in input model and it becomes impractical. So there should be coverage criteria to select minimal paths which satisfy certain conditions.

Due to this basis, the generation problem of test sequences automatically view as an optimization problem. Minimal best solution is searched by make use of Ant Colony Optimization Algorithm, a heuristic search technique. An introduction of ACO is described in section 2.4.1. In a range of search techniques we use ACO for the following reasons:

1. Well adapted for solving combinatorial optimization problems[1,11]
2. Well suited for State Based testing [11]
3. As compared to other heuristic techniques, success ratio of ACO is better [3].
4. ACO finds optimal path while traversing graph [11]

Using ACO, solutions are represented as paths generate by ants in the search space, and each ant path while traversing graph is evaluated based on fitness function. Ants that traverse the paths with highest value for the fitness function is selected as optimal solution. An overview of our automated data flow testing approach is given in figure 5.3. At an abstract level, our approach divides the process of automated data flow testing into three major steps.

1. Preparing XML of UML state machine model and is given as an input to or data flow based system. This is used to extract states, source and target transitions, invariants present within states information on different models.

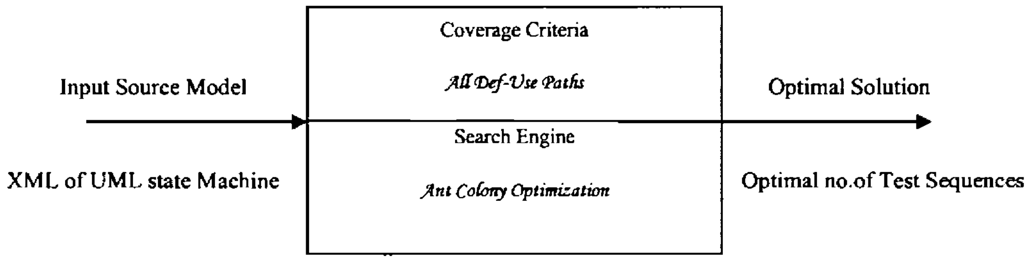


Figure 5.3 Data Flow Testing Approach View

2. Each feasible path generated from graph of the input model selected from test data that fulfill the coverage criteria using ACO.
3. Finally, all feasible test cases/paths generated from graph fulfilling coverage criteria are used to select optimal paths among them.

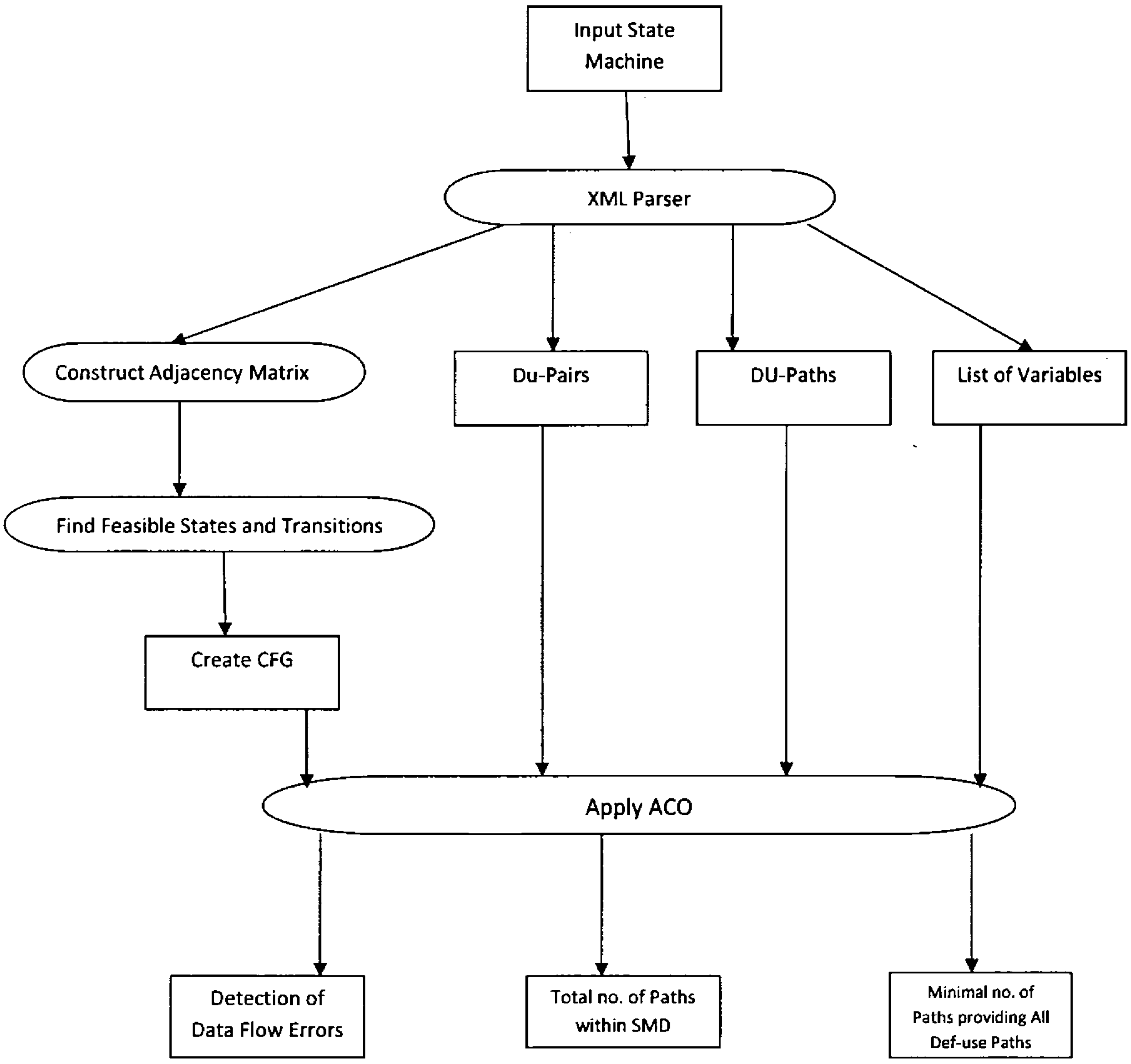

Input State Machine
XML Parser
Construct Adjacency Matrix
Du-Pairs
DU-Paths
List of Variables
Find Feasible States and Transitions
Create CFG
Apply ACO
Detection of Data Flow Errors
Total no. of Paths within SMD
Minimal no. of Paths providing All Def-use Paths

Fig 5.4 Process Flow of Proposed Approach

5.4 Test Data Selection

A quality test data is selected that is based on certain coverage it provides. All def-use paths is used as coverage criteria, a strongest criteria defined by Rapps and weyuker. More effective test cases are selected that fulfills some coverage criteria. Assignment of value to variable is known as definition while use of variable in node is use of that variable and path is a definition-use path from the point where variable definition occurs to the point it is use. All definition-use paths of each def-use pairs are covered. In case of variable is defined single time but used in more than one node than there is two def-use paths from definition to variable to each of its use. If variable is defined and then again defined in a single path before its usage than it will be error and def-use path will not from its first definition to its use because there is a killing node unbound memory location.

By using ACO, optimal solution is generated that fulfills the coverage criterion. Our approach identifies data flow relationships among variables. However to illustrate our proposed approach, we make use of state machine models as an example. Our choice of using this model is that it describes the dynamic behavior of objects. Moreover rather than automated generation of test cases based on coverage criteria, we also perform mutation testing to analyze the effectiveness of our proposed approach and to detect the seeded faults to thoroughly test the system.

5.4.1 UML State Machine Model

From the given input model of UML state machine diagram, states with invariants relationships, associated source and target transitions are extracted. For example consider the state machine model of telephone system is shown in figure 5.5. It has 8 simple states and 2 initial and final states with 16 transitions that change the state of object. Transitions are dependent on its source and target states. Adjacency matrix is created to show the connection between the states, overall rows and columns are equal to the total number of nodes within input model.

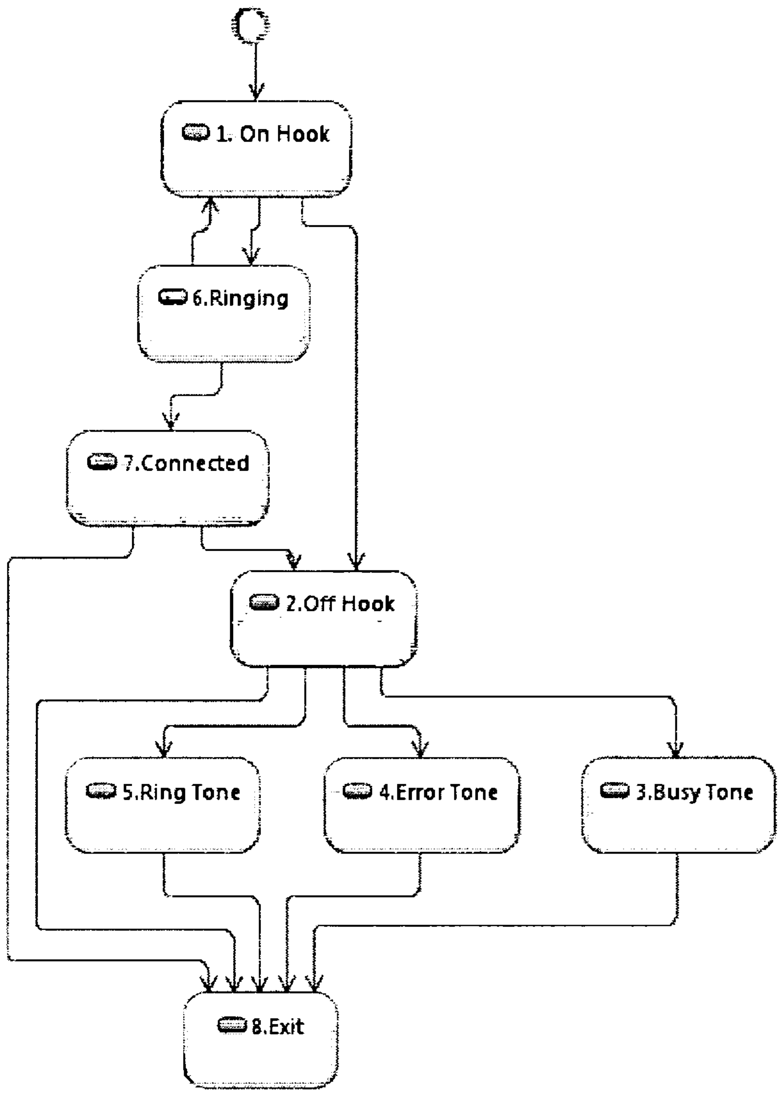


Figure 5.5 State Machine Diagram of Telephone System

$$\begin{array}{cc} & \begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} & \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Figure 5.6 Adjacency Matrix

Adjacency matrix as shown in figure 5.6 is created to show the feasible connections within a state machine diagram. Effort is reduced by using these feasible connections of paths instead of finding in the whole state machine to inspect dependencies of data. Rows and columns within matrix are equal to total states within input model. Entry is either 0 or 1. The entry 0 represents, there is no connection between states. An entity with 1 portray that there is path between states and are reachable. It keeps the information of control flow between states.

5.5 ACO Adaptation for Automated Data Flow Testing

Our approach finds an optimal solution for each input model by analyzing data flow information. ACO represent solution as number of ants deploy in each generations while traversing graph. The task of ACO is to search for optimize set of test cases covering maximum states and def-use pairs. Redundant test cases are avoided and looping problem in state machine diagram is handled by not allowing ants to revisit state more than once. For the application of search techniques to specific problem, it is necessary to represent the optimal solution, specify the objective in terms of fitness function for the evaluation of quality of searched solution. The next section explains the adaptation of ACO to our automatic data flow test problem.

5.5.1 Representation of Solution

Using ACO, overall solution is represented as ants in the search space. These ant's moves in the search space for finding best/optimal solution. The dimension of search space are considered as the graph of the input model which is created from adjacency matrix containing number of rows and columns representing connection of states in the input model. This implies that search space dimension equal to total states within input model. For example, adjacency matrix of state machine diagram of figure 5.5 contains 8 states with 2 initial and final states and 16 transitions contain 10 rows and 10 columns.

The total number of nodes in the graph will be numbered from 1 to n, n being the total number of nodes. These nodes include the possible def-use information of variables. Each of the input source models will be associated with a def-use pairs information of variables exists within states. While the transition of the input model shows the number of possible paths within graph that the ant can traverse.

This solution is implemented as a number of feasible test cases generated covering all def-use paths. A variable defined can have multiple uses in present in different nodes. For example, if we consider the definition-use pairs of state model of figure 5.5 has 11 def-use pairs. Table 5.1 shows a definition-use pairs and definition-use paths of single test case that contains total 4 variables with 6 def-use pairs covered by that test case. In State 1, 4 variables are defined and used at different states. Variable time is defined in state 1 but used at three places state 2 and state 6 and state 7, so it becomes three def-use pairs, as (1,6)(1,2)(1,7) and Def-use paths are (1->6),(1->6->7->2) and (1->6->7). This means there are three paths that must be counted to cover all definition-use paths of this def-use pair. And variable bill is defined in state 1 but used at state 7 so there is 1 definition-use pair and (1,7) and one def-use path(1-> 6->7) to cover this definition-use pair and so on.

```
Initial -> 1 -> 6 -> 7 -> 2 -> 8 -> Final

        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

        TOTAL VARIABLES USED IN THIS TRAIL

        bill , cr , rt , time , a

        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

        DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

        Variable : bill

        ( 1,7 ):          1 -> 6 -> 7

        Variable : cr

        ( 1,2 ):          1 -> 6 -> 7 -> 2

        Variable : rt

        ( 1,6 ):          1 -> 6

        Variable : time

        ( 1,6 ):          1 -> 6
        ( 1,7 ):          1 -> 6 -> 7
        ( 1,2 ):          1 -> 6 -> 7 -> 2

DATA FLOW ERRORS IN THIS TRAIL

        Variable: a
        Type of Error: UN USED
        Description: Define on nodes 8, but not used.
```

Table 5.1 def-use pairs and def-clear paths

5.5.2 Evaluation of Optimal Solution

- Selection of node while traversing

The user defined objective function evaluates the quality of data flow testing solution produced by ACO. In data flow testing, fitness value of ant indicates the aptness of the optimal test cases selection for coverage of def-use pair of their correspondence input model. For our problem of data flow testing from UML state machine models, we have defined the following fitness function:

$$PV = \frac{DU}{TN} \quad \text{..........................} \quad (1)$$

PV = Pheromones Value

DU = Total Definition-Use Pairs covered by a path

TN = Total Number of Nodes in Input Model

Fitness function is used to assess the data flow testing solution of any input model. As the fitness function contains two parts def-use pairs and total number nodes covered in that specific path. The numbers of def-use pairs are counted against each path traverse by the ant to update its pheromone value.

Using this fitness function, first ant check the number of variables covered in path that is traversed by it. Then check the definition-use pairs and definition-use paths covered in paths that are traverse by ants. Ants also check the data flow errors in those paths. For example ants check that whether variable is double defined in a single path before its usage then it detects the error. After counting def-use pairs, total number of nodes traverse by ants in each path is counted. By dividing def-use pairs counted by total number of nodes value is calculated of pheromone.

In calculating the value of pheromone, instead of using all def-use pairs within a graph, only def-use pairs are counted that are covered by a specific path. Because there a number of feasible paths within a graph, and each path cover different number of def-use pairs covering different number of nodes while restricting traversing of redundant paths. Feasible paths are those paths that are complete paths start from initial node to final node of input model. To explain this point; we will use an example of state machine model of figure 5.5. Consider the feasible paths of the input model are:

Initial -› 1 -› 6 -› 7 -› 8 -› Final

Initial -› 1 -› 6 -› 7 -› 2 -› 4 -› 8 -› Final

Initial -› 1 -› 6 -› 7 -› 2 -› 8 -› Final

Initial -› 1 -› 2 -› 3 -› 8 -› Final

Initial -› 1 -› 6 -› 7 -› 2 -› 3 -› 8 -› Final

Initial -› 1 -› 6 -› 7 -› 2 -› 5 -› 8 -› Final

Initial -› 1 -› 2 -› 8 -› Final

Initial -› 1 -› 2 -› 4 -› 8 -› Final

Initial -› 1 -› 2 -› 5 -› 7 -› 8 -› Final

Initial -› 1 -› 2 -› 5 -› 8 -› Final

In this example, there are 10 feasible test cases or paths of the input model. Consider a single test case 2;

Initial -› 1 -› 6 -› 7 -› 2 -› 4 -› 8 -› Final

In this path/test case, total numbers of variable used are 4. And 5 definition-use pairs are covered by this path. One variable is defined at single place at node 1 while used at 3 places, in node 6, 2 and 7. So it becomes 3 def-use pairs of single variable and there is 1 definition-use path to cover each def-use pair. Another variable rt is defined at node 1 but used at 2 places in model, at state 4 and 6. So there are 2 def-use pairs and def-clear path. Variable that is defined and used single time in a node so it has 1 definition-use pair and has a single definition-use path to traverse it. So there total numbers of def-use pairs are 5 and total of nodes traversed in this specific path are 6. Its pheromone value becomes 0.83.

The nodes traverses in this path are unique, revisiting of nodes is not allowed due to looping problem. In each path, each node is visited once a time. But in different paths, a single node may be covered many times. Redundant test cases are also avoided to save time. When a single node is covered many times in different paths, each time its pheromone value will be increased. Its means node is traverse by more number of ants having highest probability and its pheromone value will be increases. For example in test cases as defined above; node1, 6, 7, and 8 is traverse by many ants in their path. Consider the node 7 which is traverse 6 times by ants in 6 feasible test cases. When it is traverse first time by ants, pheromone value set by ants is 0.5 in a first feasible test case. When node 7 is traverse 2nd time by ants, its more pheromone will be added to this node by other ants traversing this. Consider the first test sequence in which 3 def-use pairs are covered having pheromone value 0.75, and then updated pheromone value will be 0.83+0.75 = 1.58. First value is pheromone value while traversing this node first time and 2nd value is pheromone value set by ants in other paths, so its pheromone value is updated and so on.

Ants sense the pheromone traces at each current point in graph of its directly connected nodes and leave pheromone traces while traversing that node. Pheromone value is set by ants after completing its tour because if ants set the pheromone value of nodes before reaching to its destination and will not reach to its destination then other ants also follow this path and all of them don't reach to destination. As ants make their decision based on this value, this value is set by ants after completing their tour. And the pheromones value is updated each time it is traverse, nodes with highest pheromone will be highlighted by more traversal of ants.

Nodes that are traversed many times also have highest probability. Because probability depends on pheromone value and heuristic factor. Alpha and beta are constants, set their value 1. Visited status of each node is tracked. Value 1 of state/node means that node is not visited yet. If node is visited than its value changes to 0. Based on these value ants don't visit node twice. During tour, each time ants made to visit node it also checked the visited status of node.

When ants are made to traverse graph, they check these value at each point where they are standing and want to move to visit next node. If ant finds nodes with same probability in their path that is directly connected to current node, then it checks the desirability value of node. If both nodes are not visited yet than it make decision randomly and if node visited status is 0 that means it is already visited and second node visited status is 1 it means it is not visited. So ant make a decision to move to that node which is not visited having same probability.

To avoid the problem of ants, stuck in local optima is avoided by setting the evaporation rate of nodes. Pheromone value of each node is constantly evaporated. Because if this value is not considered, pheromone value on each node remain same and ants will follow the same path in each generations. Nodes that have highest pheromone value have low evaporation rate than those paths that have low pheromone value.

Mutation testing is performed to analyze the effectiveness of model. Data flow faults are identified from literature which are seeded into models and are detected by ants the type of fault and its location is identified.

5.5.3 Deriving an Optimal Solution

In first generation of ACO, ants are made to move randomly for selecting the paths. Because at start, probability of each node is zero due to each node pheromone value is set zero. Pheromone value is zero as ants set this value after completing its first tour. After first generation ants set the pheromone value of each node of every path that is used by ants of next generation. And optimal solution is selected in each generation by calculating probability value of each path. Probability value ranges from 0 to 1.

In ACO, each ant while traversing graph is associated with four factors. In each decision of ant to traverse a certain node, it keeps record of their current position in a graph, visible connection of nodes from current position, heuristic factor, and pheromone value. Current position of ant is its position in graph when it wants to make decision, while after making decision its current position will be changed and new position will be its current position.

Fitness function is used to evaluate the ant's decisions that decide the quality of the data flow testing solution. Paths having high fitness value are stored in optimal test case that is best optimal solution in a graph. Fitness value of each optimal test case is stored and optimal numbers of test cases are selected from a number of feasible test cases. On each iteration, fitness value is calculated and stored to make comparison with later generations of ants to select an optimal solution. In each generation, fitness value of each iteration is compared, if the current values are better than existing iteration than optimal paths will be updated. The probability of optimal paths in each generation is calculated using the formula in (2).

$$P = \{(\tau ij)^{\alpha} * (\eta ij)^{\beta}\} / \Sigma (\tau ij)^{\alpha} * (\eta ij)^{\beta} \ldots\ldots\ldots\ldots (2)$$

P = Probability

Alpha & Beta are constants

τ = Pheromones Value

η = Heuristic function

Optimal paths are selected from feasible paths of the given input model. Optimal paths are those paths that cover maximum def-use pairs. Minimal numbers of paths are selected that cover maximum states as well as def-use pairs while ensuring complete detection of data flow errors result in reducing the search space. In each generation, optimal paths are different due to coverage of different number of def-use pairs and states.

$$\tau ij = (1-r)\, \tau ij + \Delta\tau ij \ldots\ldots\ldots\ldots (3)$$

Where r= Rate of evaporation of pheromones between 0 and 1.

$\Delta\tau ij$= Total amount of pheromones set down by ants when it traverse from edge i to j.

Using formula 3, pheromones of each path evaporates. Path covering large number of definition-use pairs have low evaporation rate than the path covering low number of definition-use pair in which pheromone value evaporate quickly. But it shows better results when r is between 0.25 to 0.35.

5.5.4 Parameter Tuning

In the searching of an optimal solution by ACO, parameters play an important role. In our approach, following parameter values are set to find an optimal solution:

1. First parameter is number of ants, for example number of ants sends for exploring the search space. In automatic setting we set the number of ants to 100.
2. Evaporation coefficient that changes the decision of ants in each iteration.
3. We set the maximum number of generations to 10.

5.6 Automated Data Flow Testing

By using the proposed approach, automatic process of data flow testing is divided into six main steps which are shown in figure 5.5.

1. Prepare Input Source Model.
2. Create Control Flow Graph.
3. Identify data Flow Information.
4. Automatically generate feasible test cases.
5. Optimal set of test cases using Optimal Solution.
6. Data flow Error information against each path.

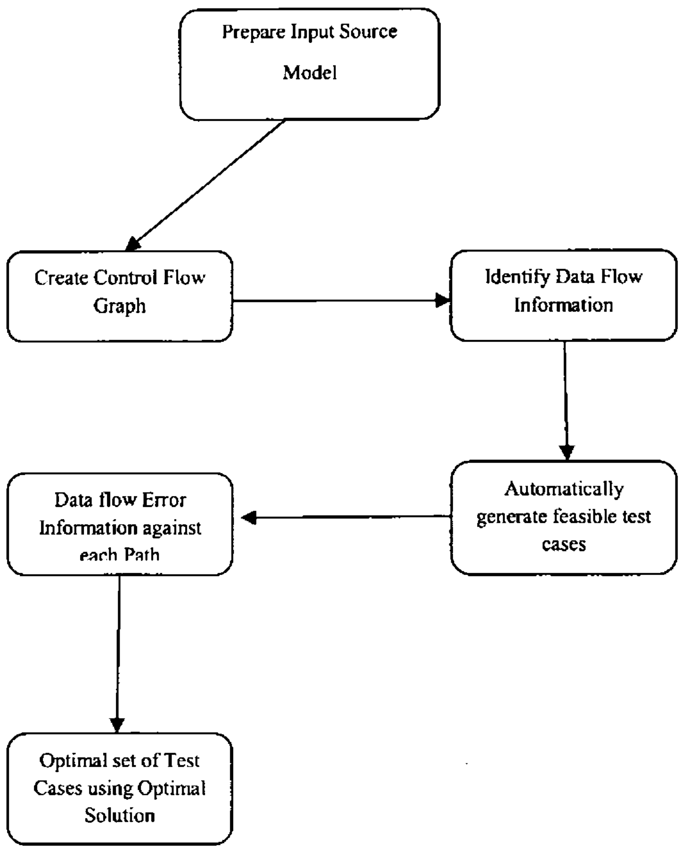

Prepare Input Source Model
Create Control Flow Graph
Identify Data Flow Information
Automatically generate feasible test cases
Data flow Error Information against each Path
Optimal set of Test Cases using Optimal Solution

Figure 5.7 Steps of Automated Data Flow Testing Process

5.6.1 Prepare the Input Source Model

In our approach, input model to the tool is given in the form of XML. Input is prepared by generating the XML using UML modeling tool enterprise architect, a comprehensive UML designed tool.

5.6.2 Create Control Flow Graph

In our proposed approach, after selecting source model it is converted into adjacency matrix. Control flow information is collected by creating a graph from adjacency matrix in which states and transitions information shown is extracted from XML of input model. XML of each input model is passed to collect its control and data flow information regarding model. Control flow graph shows the flow of control between states and number of possible paths from initial state to its final states. Total number of states and transitions in a control-flow graph is equal to the rows and columns of adjacency matrix. This information is used by automatic code generator for the generation of set of feasible test cases against XML of each input model.

5.6.3 Identify data Flow Information

Data flow information exists within states/nodes of input model are extracted by parser. This information is necessary for analysis of data and data flow testing. Our approach is generic and can be used to perform data flow testing of any input model. Test data is selected that provide the coverage of all def-use paths.

5.6.4 Automatically generate feasible test cases

After creating a control flow graph, appropriate test data are selected to provide coverage of all def-use paths in a search space. Search engine search for the feasible paths among set of possible paths in each input model. Ant Colony Algorithm is used for searching the entire search space. The task of ACO is to only select the complete paths starting from initial state to its final state. Each path traverse by ants in each generation is evaluated on the basis of fitness function while not allowing the redundant paths to be generated.

5.6.5 Data flow Error information against each path

Mutation testing is performed to see the effectiveness of algorithm. Errors are detected against each path and position of errors and type of errors are identified within search space.

5.6.6 Optimum set of test cases using Optimal Solution

Finally, the optimal solution is selected based on solution search by the ACO. Minimal numbers of test cases are selected for providing maximum all def-use paths and state coverage.

Chapter 6

TOOL IMPLEMENTATION

6.1 Introduction

Our proposed approach is implemented in a tool that is named as data flow generator. Tool is capable of generating optimal number of test sequences using data flow information from UML state machine. Our motivation to select this model as UML state machines represents the dynamic system behavior. Moreover, invariants within states are considered to ensure correct data flow. Tool is capable or efficient enough in detection of data flow errors.

This chapter explains the tool that is based on our approach. Section 6.2 describes the tool architecture and its main workings. Specific details related to implementation are given in section 6.3. Section 6.4 describes the process flow of our tool.

6.2 Tool Architecture

Figure 6.1 shows architecture of our tool. Architecture of tool has three main components, XML Parser, Search Engine, and Data Flow Testing. Our tool takes a source model as an input. Input is handled by the XML Parser. Optimal solution is found by the search engine for the given input models. Finally the data flow testing is performed by using the optimal solution performed by the search engine and test cases are generated to cover these def-use pairs. Moreover tool is capable of detecting faults seeded by mutation operator. Explanations of three major components are as follows.

6.2.1 XML Parser

XML parser initiates the execution of tool. First it takes an input model of state machine diagram to extract state, transition and invariant information from model. The training data are XML files of state machine diagrams. Main function of XML parser is listed below;

- Load the whole XML file
- Extract the states and transitions tags
- Extract invariant's within states
- Create Adjacency Matrix to show feasible connection between states.

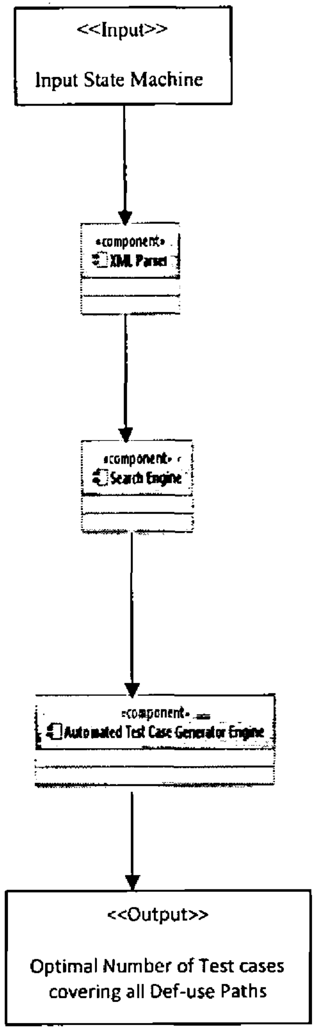


Figure 6.1 Tool Architecture

Tool takes a state machine diagram as an input stored in input file. Stat machine describe the dynamic behavior of object containing invariants within states. XML parser organizes the source, target state and data flow information that are used by next components. For the input source models, following tasks are performed by xml parser.

- Store the input model.
- Count the number of states and transitions.

- Extract data flow information.

6.2.2 Search Engine

Search engine is main component of tool. ACO algorithm is used as the search engine. Major role of search engine is to find best/optimal solution for any input model. Optimal solution is in the form of optimal test sequences completing the covering coverage criteria. One test case shows the total number of covered definition-use pairs and data flow errors.

Search engine is initializes by creating the control flow graph for traversing the graph. Quality of solution is evaluated against fitness function which is defined in section 5.5.2. The fitness values calculated by ants, ACO parameters are updated and many other optimal solutions are generated in next generations. Component of the search engine active until total generations of ACO is completed. Final optimal solution is selected based on having maximum fitness value.

6.2.3 Automated test case generator Engine

This is an important component of our tool. Basic purpose of automated test case generator is the production of feasible and minimal number of test sequences providing maximum definition-use pair coverage, related to input model. It does by using optimal solution generated by search engine component.

Automated test case generator takes input of optimal solution generated by search engine. For every input model, it searches the optimal test cases. Def-use pair's information is then used to produce the output. In this way, total number of definition-use pairs covered in test case is produced as an output in automated test case generator engine for the input source model.

This component also detects the data flow error present within states of input model. The automatic detection of data flow errors reduces the time and cost consumption in manual effort.

6.3 Tool Implementation

Tool is implemented using .net language. We use Microsoft visual studio 2008 for tool implementation. From implementation point of view, tool is organized into eight classes shown in class diagram in figure 6.2.Path is the major class of our tool that select the optimum paths from all the feasible test sequences of input model. ACO class contains heuristic search optimization technique. Editor class mainly deals with the GUI of tool. Class of variables stores variables information that is present within states. XML parser class deals with loading of XML file and extraction of specific information from XML file. Graph class creates the adjacency matrix and control flow graph and graph is updated as the information change. Class of error encapsulates the data flow errors information. Node class shows the states and transition information. A description of the main classes of tool is given below.

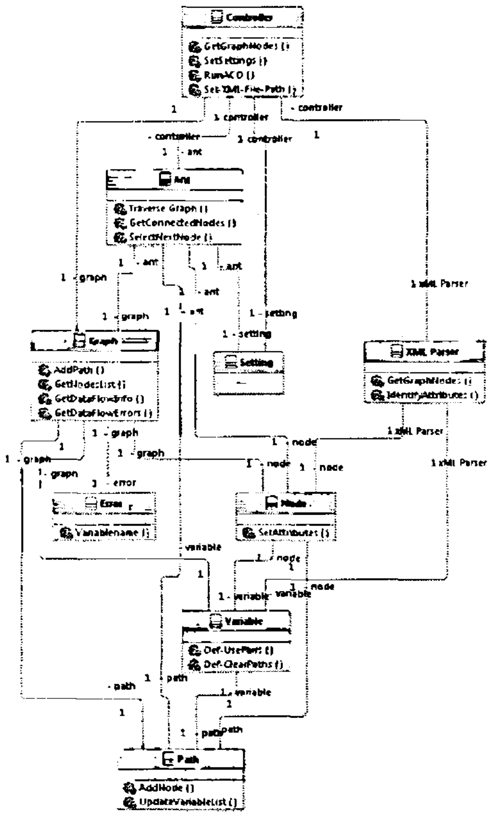


Figure 6.2 Class Diagram

6.3.1 Path

Path class is responsible for managing the paths traverse by ants in a number of generations and produce automatically the minimal test sequences. Specifically following tasks are performed in this class.

1. Manage the graph nodes and transition information.
2. Manage the information of total number of nodes within a model.
3. Manage the updation of pheromone value on graph nodes.
4. Manage and organize the paths traverse by ants.
5. Manage and consider feasible paths in optimal solution.
6. Quality of each solution is evaluated that is generated by ACO.
7. Final optimal solution is selected.

Path class consists of five methods. These methods perform the following tasks.

AddNode

It keeps tracks of nodes traverse by ants until final node is reached and adds them to path string. And nodes that are traversed change their status into visited and change their value to 0. Because if node is not visited once its status becomes1. Also updation of pheromone and variable function is called in this class.

UpdateVariableList

This method maintains the variable updation list because as more nodes are added into track variable list is updated.

UpdatePathPheromonesValue

This contains our fitness function; pheromone value is updated in this formula.

Pheromone Value = Total definition-use pairs covered in path / Total nodes covered by that path.

Pheromone value is updated in current graph set by ants by traversing the graph. Evaporation rate of pheromone value on each node is also set in this method by a constant factor.

Visitedstatus

This method keeps tracks of information regarding visited status of nodes that are directly connected to current node.

GetSubpathsString

This path keeps track of initial and final node.

6.3.2 ACO

TraverseGraph

It set current position of ant in graph initially at state 0. It gets all nodes which are one transition apart from current node and select state with highest probability.

GetConnectedNodes

It manages the list of connected Nodes until ants reach to the final node.

SelectNextNode

This method maintains the list of nodes that ants select in traversing graph having highest probability. If current node is connected with more than one nodes with one node having highest probability than ant will select that node to traverse. But more than one node having same highest probability than ant will take decision randomly. If there is no connected node and current node is also not a final node then path will be broken and not considered in feasible paths or test cases.

CalculateProbability

Probability of node is calculated in this method using pheromone value, desirability factor, alpha and beta value. Alpha and beta are constants.

6.3.3 Graph

CompareNodePosition

This method compares the position of nodes to arrange them in sorted order.

CreateGraph

It keeps track of total number of nodes within input model. It takes input and compare graph position to create the graph.

SetGraphValue

This method manages and identifies the source and target nodes information.

AddPath

It gets information from paths methods regarding different complete paths that are traverse by ants in a number of generations. This method adds that paths to feasible paths.

GetOptimumPath

This method selects the optimum paths from feasible test cases or paths, infeasible paths are not included in optimum paths. Optimum paths are selected on the basis of maximum def-use pairs covered by those paths.

GetDataFlowInfo

This method keeps track of variable information present in nodes. By getting all the list of variable in each node from node class it categorize them as whether variable is defined or used. It stores this information and displays data flow information against each path.

GetDataFlowErrors

This method manages the data flow faults information exists in each node. It identifies error within nodes and checks whether it undefined, unused, double define error etc. It also keeps track of path and node number where error exists.

GetNodeList

It adds and manages list of nodes that are one by one selected by ant in different generation.

GetNode

This method gets the node position, visited status and def-use pair information.

GetDefUseNodeCount

It counts the total definition-use pairs.

GetTotalNodes

It count total number of nodes covered in each path.

UpdateGraph

This method manages the graph that is updated each time pheromone value is updated.

6.3.4 Variables

Status

This method gets and sets status of variable.

Name

It gets and sets the name of variable.

DefIndex

This method manages the index of define variable.

UseIndex

This method manages the index of use variable.

DefNodesList

It maintains the information about number of times a variable is defined. And redundant number of define variables list is avoided.

UseNodesList

It maintains the information about number of times a variable is used. And redundant number of used variables list is avoided.

DefUsePairs

It maintains the total number of def-use pairs within input model.

DefUsePaths

This method keeps track of def-use paths and redundant def- use path are removed.

AddDefNodeNumber

It adds the number or position of nodes where single variable is multiple times defined.

AddUseNodeNumber

It adds the number or position of node where single variable is multiple times used.

6.3.5 Editor

Editor class deals with the GUI of tool. Controller of our code (tool name) application controller.cs initiates the execution of our tool. It performs the following function;

- Manage GUI of tool name
- Manage the tool project dictionary
- File loading and reading

6.3.6 Nodes

Clone

It just return the Clone or a reference of a graph node instead of sending original node. It returns the shallow copy of original node.

SetAttributes

It checks whether state contains any variable or not.

6.3.7 Errors

ErrorType

This method stores the error exists in node.

ErrorTypeStr

It identifies the type of error exists in node.

ErrorDescriptionStr

It stores the errors details for example node where variable is defined but not used in any other node or used in specific node but not defined in any other position in input model.

VariableName

It stores the name of variable.

DefNodesList

It stores the list of defined nodes.

UseNodesList

It stores the list of used nodes.

6.3.8 XML Parser

GetGraphNodes

This method extracts the overall intermediated states known as simple states within input model and also initial and final state of the input model. It also extracts the invariants information present within each state from XML of input model. And pass that information of states to create graph method.

GetGraphTransitions

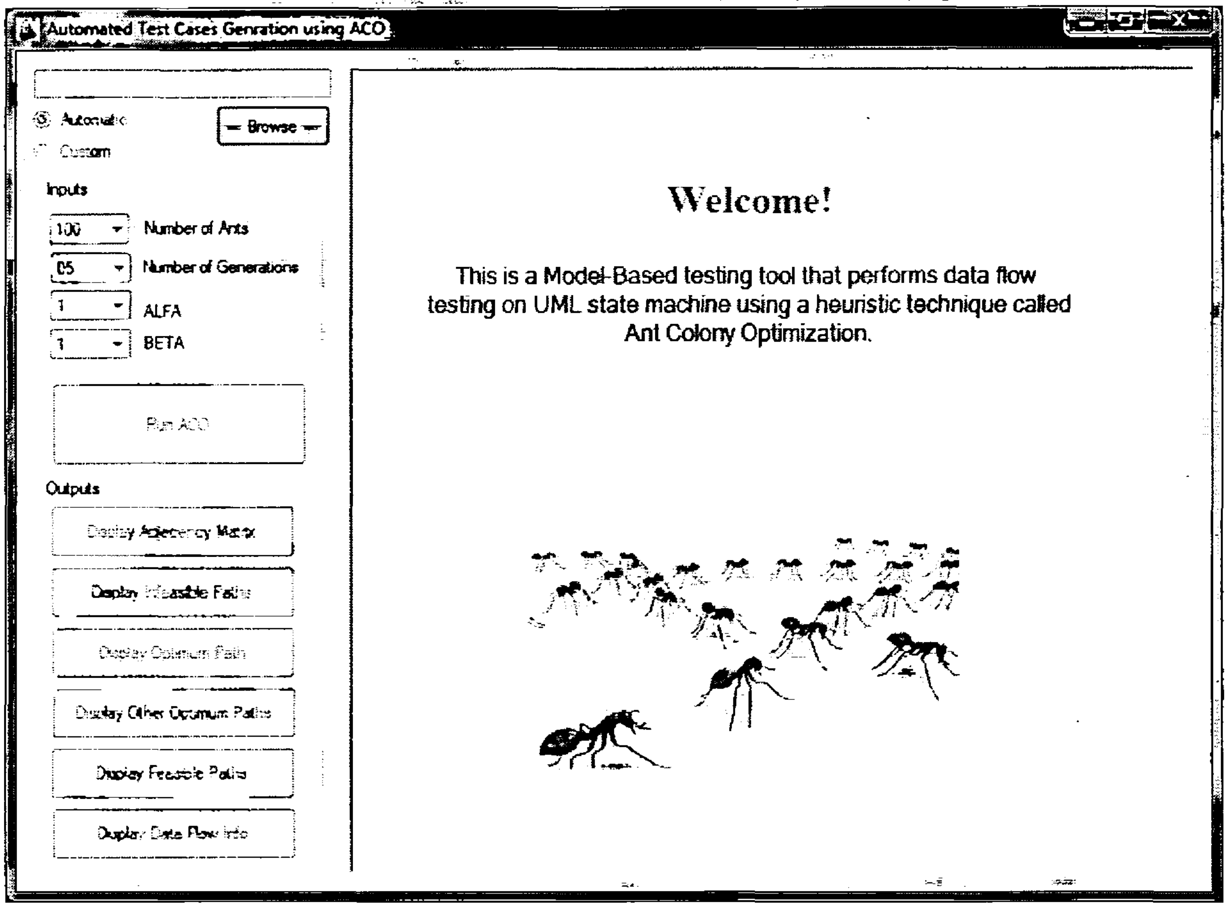
This method extracts the source and target transitions of each state from XML of input model.

IdentifyAttributes

It identifies the attributes from XML of state machine and if more than one variable are presents in one state than it sphts them and stores them separately.

6.4 Tool Process flow

Main interface of our tool is shown in Figure 6.3. It generates test sequences against each input model. The whole information of tool and its thorough user manual is discussed in Appendix. Process of data flow testing can be divided into three major steps.



100 Number of Ants
05 Number of Generations
1 ALFA
1 BETA

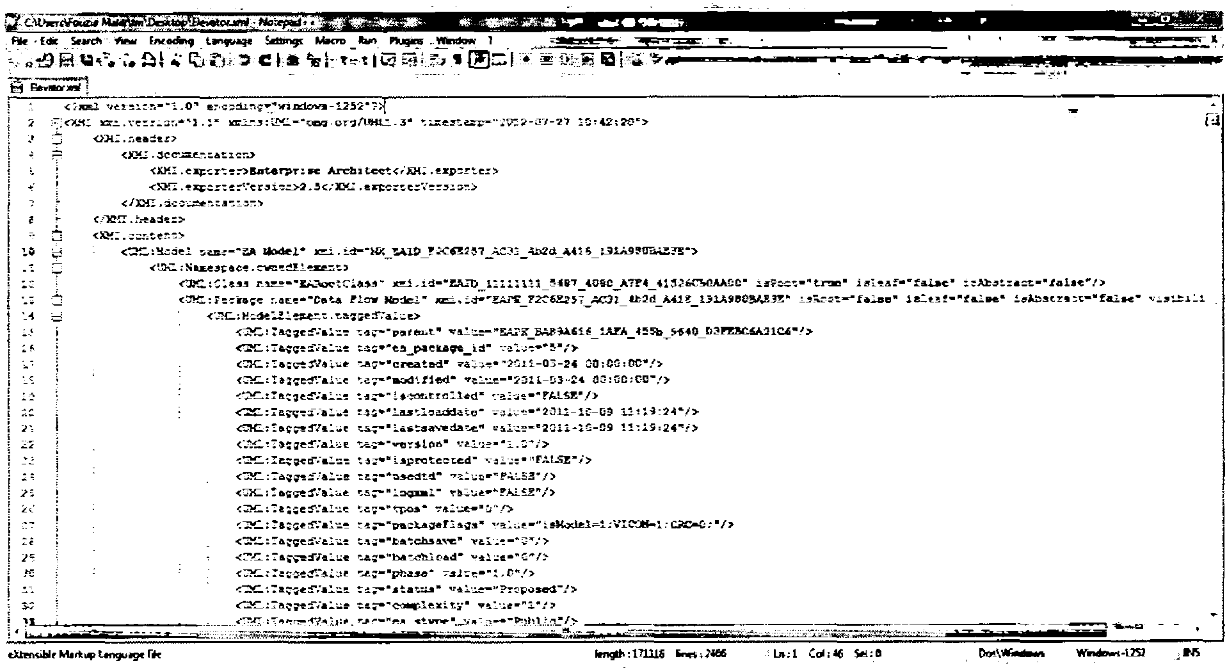
Welcome!

This is a Model-Based testing tool that performs data flow testing on UML state machine using a heuristic technique called Ant Colony Optimization.

Figure 6.3 Main Window of Tool

6.4.1 Input Source Model

After running the application, input XML file is browse from the main interface. Browse option is available in toolbar of tool. XML contains activity as well as state diagram. It contains complete information of input model states, transitions, invariants etc. Specific information related to perform data flow testing is extracted from the XML. When user browses the input XML file, it loads and stores the xml file path than extracting nodes, transitions and variable information in separate file and arranges them in sorted order according to source and target transitions. This is one by the xml-parser class. XML file is shown in fig 6.4.

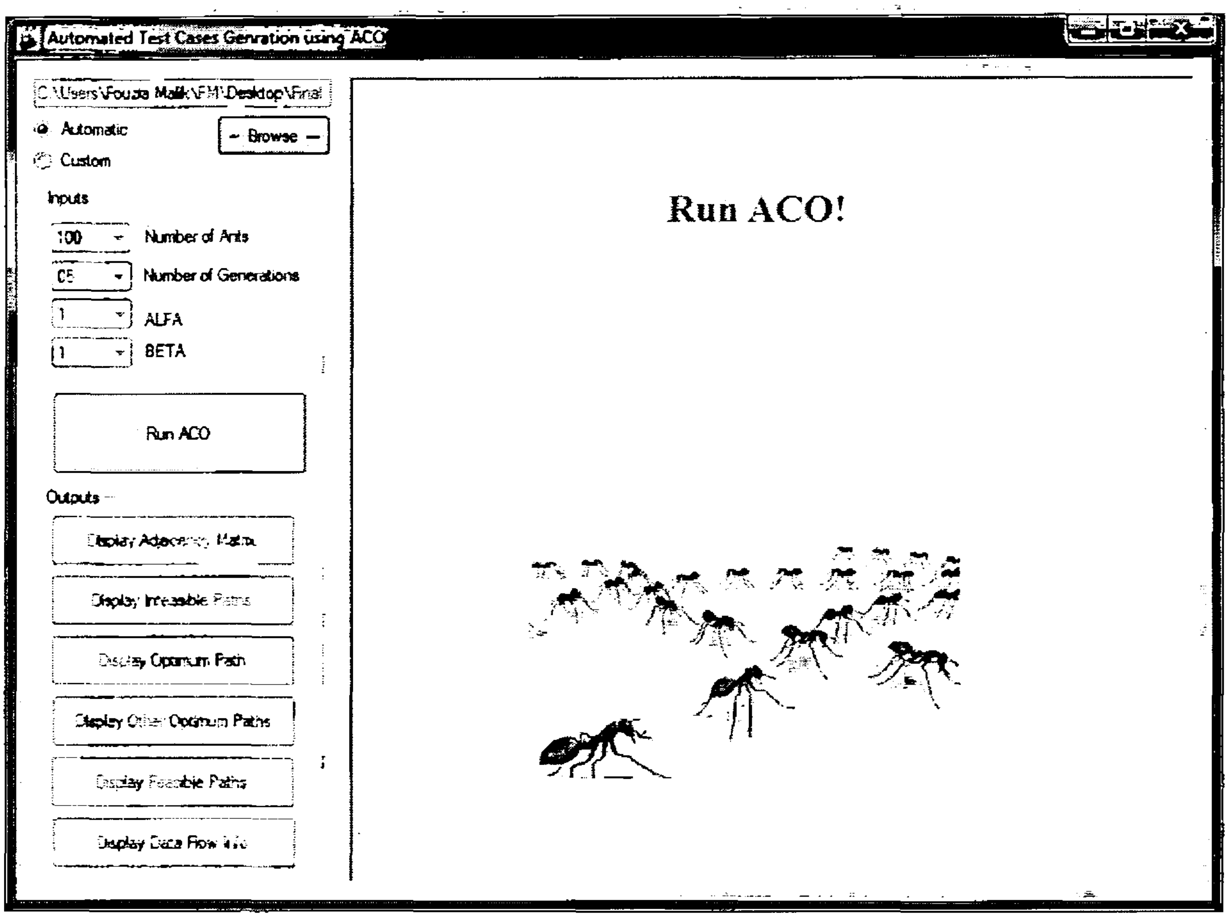

C:\Users\Fouzia Malik\Desktop\Elevator.xml - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Elevator.xml
eXtensible Markup Language file
length : 171116 lines : 2466
Ln : 1 Col : 46 Sel : 0
Dos\Windows
Windows-1252
INS

```
<?xml version="1.0" encoding="windows-1252"?>
<XMI xmi.version="1.1" xmlns:UML="omg.org/UML1.3" timestamp="2012-07-27 10:42:20">
    <XMI.header>
        <XMI.documentation>
            <XMI.exporter>Enterprise Architect</XMI.exporter>
            <XMI.exporterVersion>2.5</XMI.exporterVersion>
        </XMI.documentation>
    </XMI.header>
    <XMI.content>
        <UML:Model name="EA Model" xmi.id="MX_EAID_F2C6E257_AC31_4b2d_A416_131A980BAE3E">
            <UML:Namespace.ownedElement>
                <UML:Class name="EARootClass" xmi.id="EAID_11111111_5487_4080_A7F4_41526C50AA00" isRoot="true" isLeaf="false" isAbstract="false"/>
                <UML:Package name="Data Flow Model" xmi.id="EAPK_F2C6E257_AC31_4b2d_A416_131A980BAE3E" isRoot="false" isLeaf="false" isAbstract="false" visibili
                    <UML:ModelElement.taggedValue>
                        <UML:TaggedValue tag="parent" value="EAPK_BAB9A616_1AFA_455b_5640_D3FEBC6A21C6"/>
                        <UML:TaggedValue tag="ea_package_id" value="5"/>
                        <UML:TaggedValue tag="created" value="2011-03-24 00:00:00"/>
                        <UML:TaggedValue tag="modified" value="2011-03-24 00:00:00"/>
                        <UML:TaggedValue tag="iscontrolled" value="FALSE"/>
                        <UML:TaggedValue tag="lastloaddate" value="2012-10-09 11:19:24"/>
                        <UML:TaggedValue tag="lastsavedate" value="2012-10-09 11:19:24"/>
                        <UML:TaggedValue tag="version" value="1.0"/>
                        <UML:TaggedValue tag="isprotected" value="FALSE"/>
                        <UML:TaggedValue tag="usedtd" value="FALSE"/>
                        <UML:TaggedValue tag="logxml" value="FALSE"/>
                        <UML:TaggedValue tag="tpos" value="0"/>
                        <UML:TaggedValue tag="packageflags" value="isModel=1;VICON=1;CRC=0;"/>
                        <UML:TaggedValue tag="batchsave" value="0"/>
                        <UML:TaggedValue tag="batchload" value="0"/>
                        <UML:TaggedValue tag="phase" value="1.0"/>
                        <UML:TaggedValue tag="status" value="Proposed"/>
                        <UML:TaggedValue tag="complexity" value="1"/>
                        <UML:TaggedValue tag="ea_stype" value="Public"/>
```

Figure 6.4 XML File of Input Model of ECS

6.4.2 Parameter Setting

After the input file is loaded, parameters to perform testing are set automatic or custom. In automatic specific number of generations and ants are selected for traversing. In custom setting use selects the settings as under.



● Automatic
○ Custom

Inputs

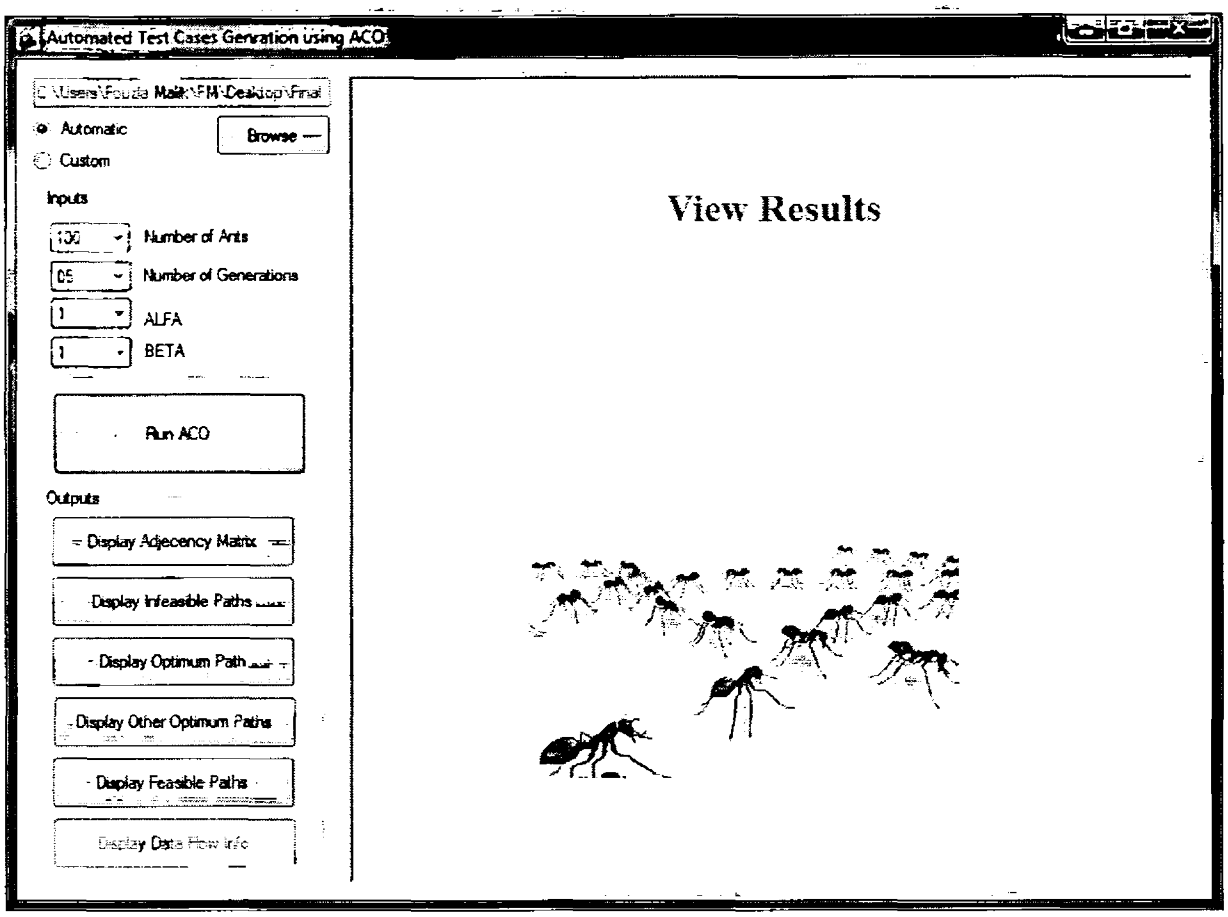
100 Number of Ants
C5 Number of Generations
1 ALFA
1 BETA

Figure 6.5: Main GUI (Select Parameters)

6.4.3 Run ACO

The third option is user select the RUN ACO option to perform data flow testing producing the desired results. It important step because using the ACO algorithm, optimal solution is generated. Furthermore, this solution shows the complete coverage of data flow coverage criteria. Majority of classes are associated and functions are performed against this button.

The target test cases and data flow information is stored and displayed according to their relevant buttons that just displays the output. Each output is calculated and stored in its relevant class. Figure 6.6 shows the output produced for the input model of Elevator State Machine Diagram".



C:\Users\Fouzia Malik\FM\Desktop\Final

[✓ Automatic] Custom

100 Number of Ants

05 Number of Generations

1 ALFA

1 BETA

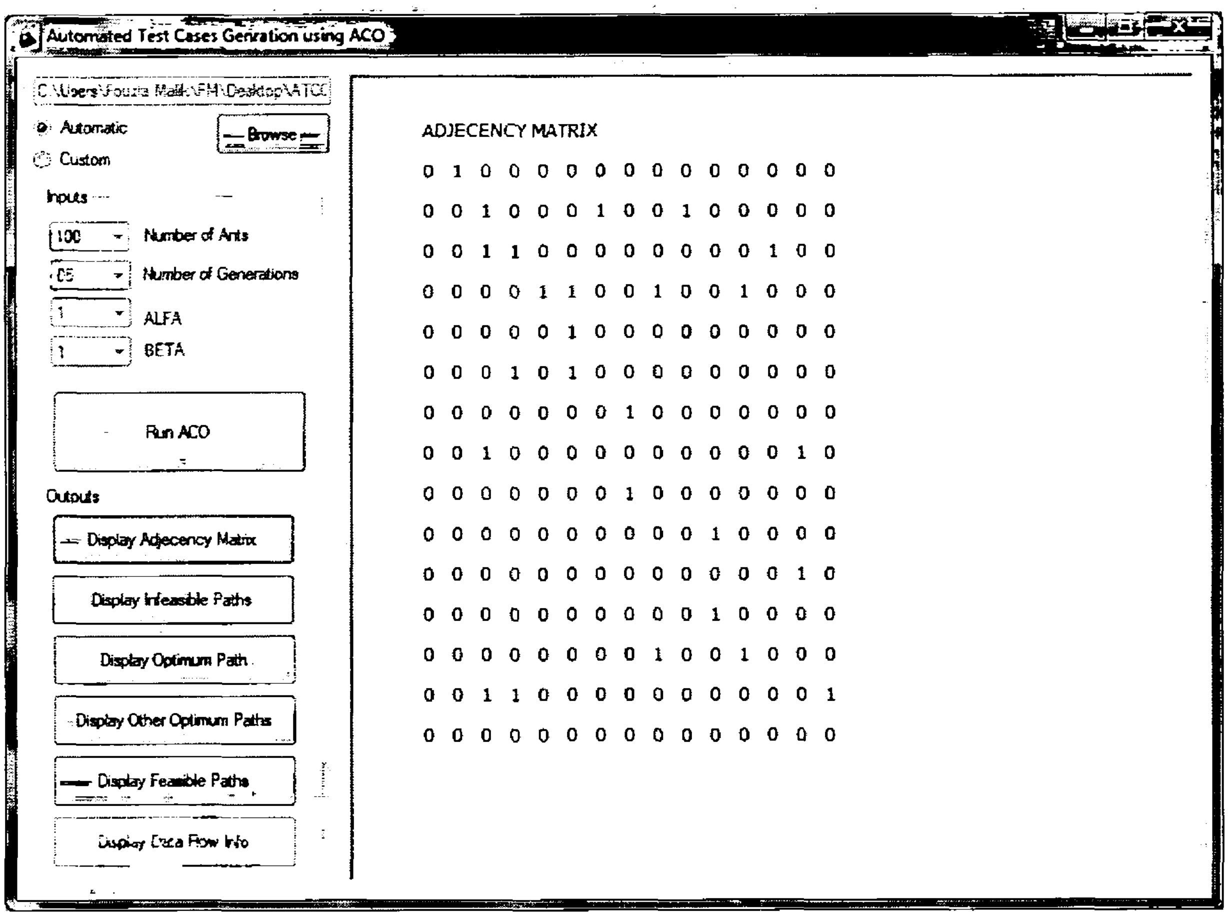
Figure 6.6: Main GUI (Select Run ACO option to Generate Results)

Finally the user select the outputs to show generated by ACO. Output generated by tool is shown according to available options in main interface. Class of ACO-GUI.cs is mainly responsible for performing this task. Output contains the following information.

1. Adjacency Matrix showing feasible connection between states
2. Total number of feasible paths in input model
3. Optimal number of test sequences covering maximum def-use pairs
4. Total number of variables presents in input model
5. Total definition-use pairs covered by each path
6. Total number of errors exists in each node of each path
7. Details of the type of error exists in nodes of each paths
8. Identifies the error location within node of each path

Figure 6.7 shows the adjacency matrix for creating the graph. Feasible test cases or paths within input model are shown in figure 6.8. Optimal number of test cases covering def-use pairs information as well error exists in each optimal path is shown in figure 6.9, 6.10, 6.11,6.12, 6.13, and 6.14.

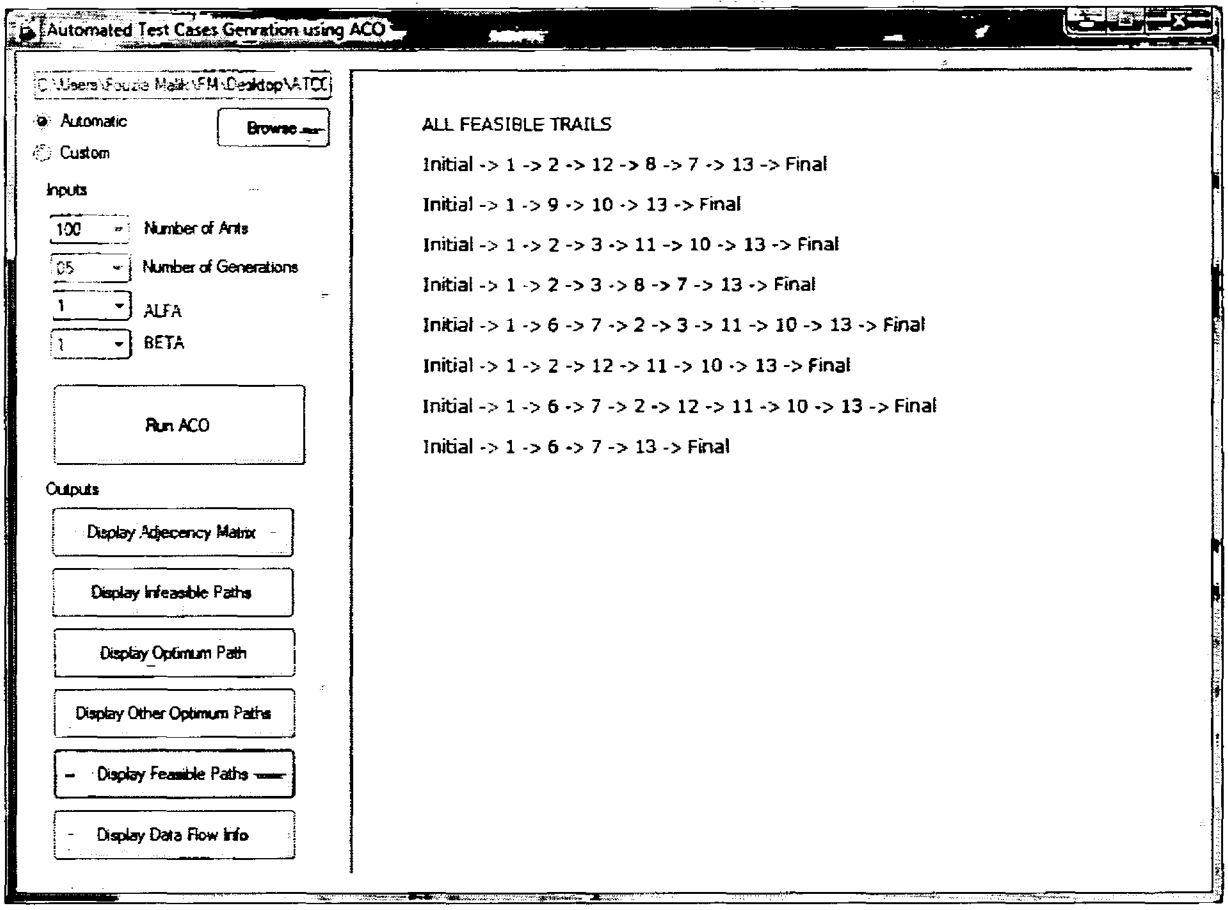
6.4.3.1 Create Adjacency Matrix



```
ADJECENCY MATRIX
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 1 0 0 1 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 1 1 0 0 1 0 0 1 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 1 0 0 0
0 0 1 1 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 6.7: Adjacency Matrix of ECS Model

6.4.3.2 Generate Feasible Test sequences



Automatic [✓]
Custom

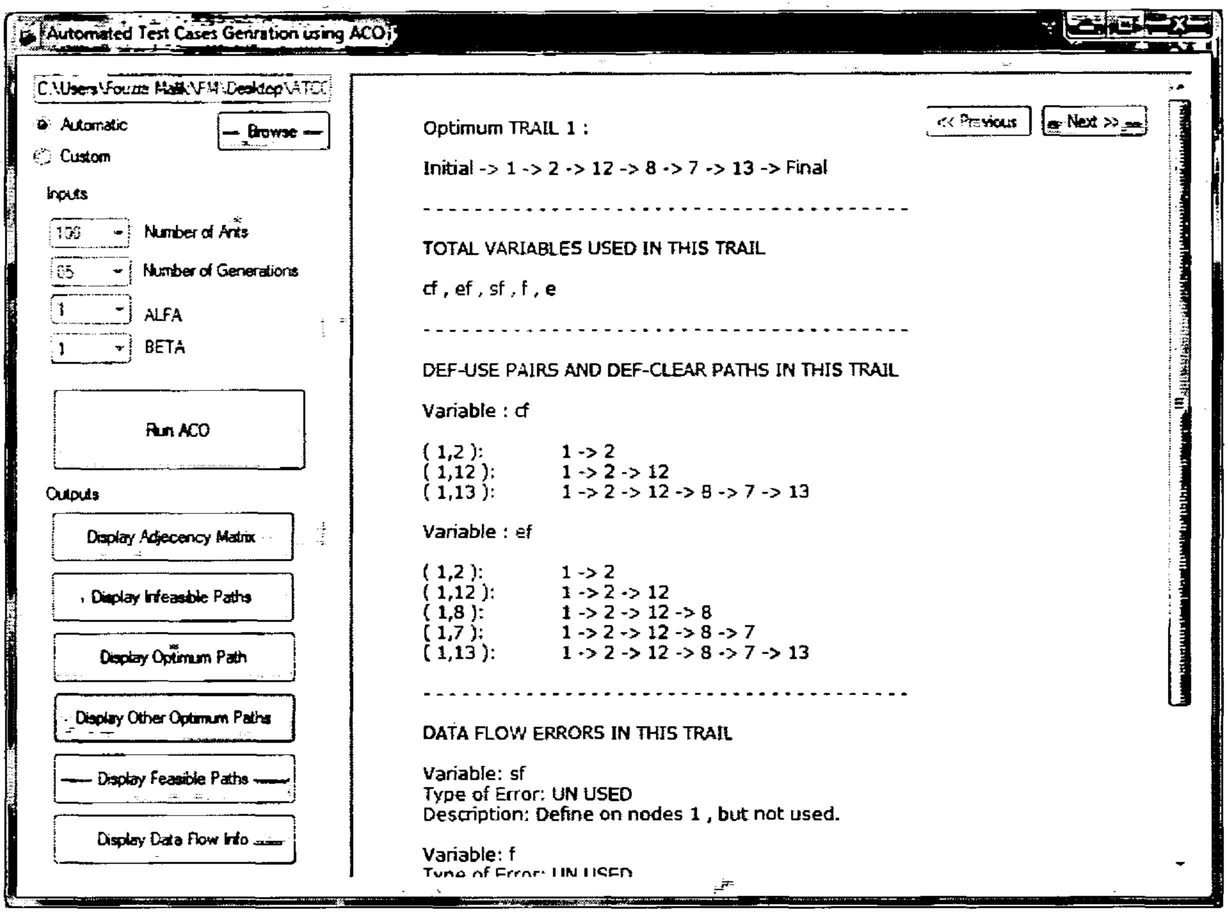
Inputs

100 Number of Ants
05 Number of Generations
1 ALFA
1 BETA

```
ALL FEASIBLE TRAILS
Initial -> 1 -> 2 -> 12 -> 8 -> 7 -> 13 -> Final
Initial -> 1 -> 9 -> 10 -> 13 -> Final
Initial -> 1 -> 2 -> 3 -> 11 -> 10 -> 13 -> Final
Initial -> 1 -> 2 -> 3 -> 8 -> 7 -> 13 -> Final
Initial -> 1 -> 6 -> 7 -> 2 -> 3 -> 11 -> 10 -> 13 -> Final
Initial -> 1 -> 2 -> 12 -> 11 -> 10 -> 13 -> Final
Initial -> 1 -> 6 -> 7 -> 2 -> 12 -> 11 -> 10 -> 13 -> Final
Initial -> 1 -> 6 -> 7 -> 13 -> Final
```

Figure 6.8: Feasible Test cases of Input Model of ECS

6.4.3.3 Generate Optimal Test Sequences

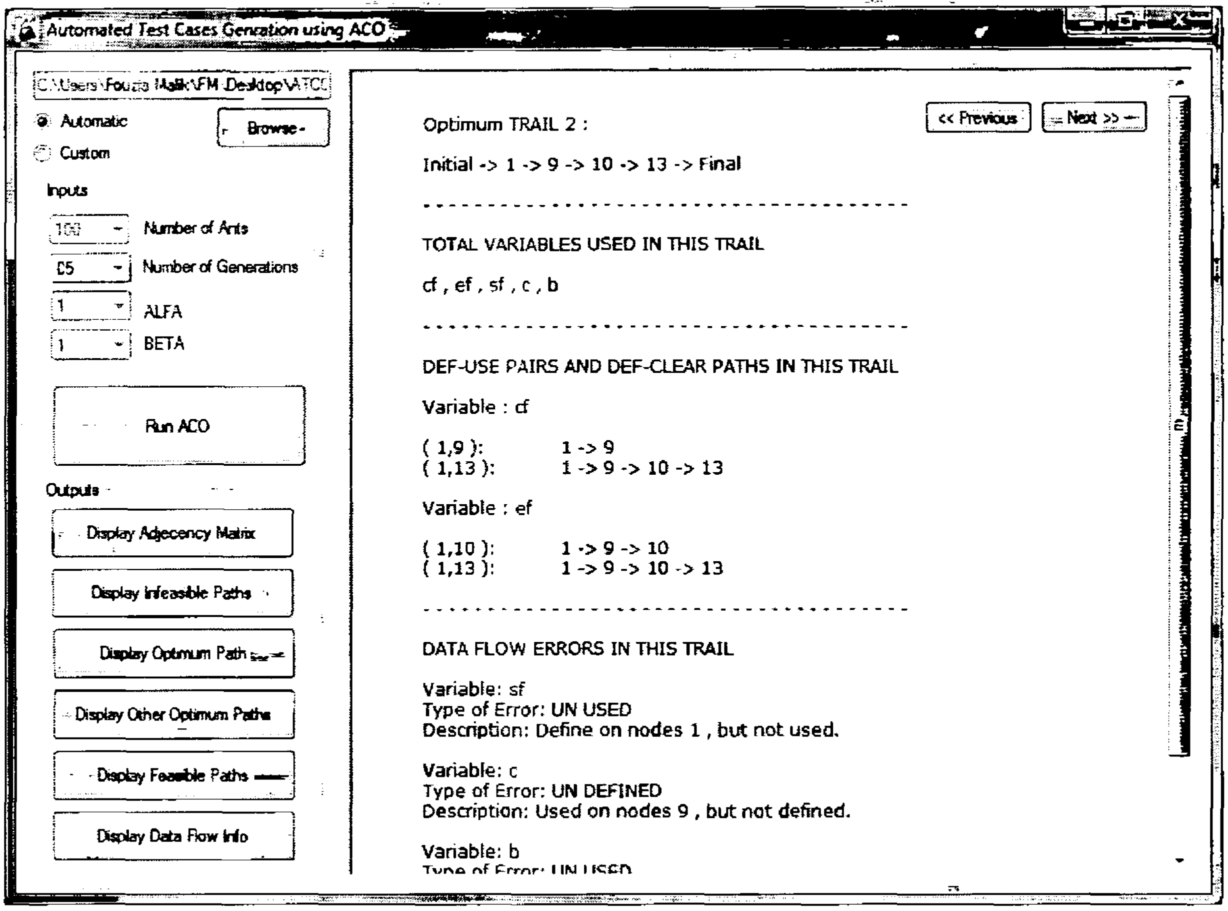


Number of Ants: 100
Number of Generations: 05
ALFA: 1
BETA: 1

```
Optimum TRAIL 1 :

Initial -> 1 -> 2 -> 12 -> 8 -> 7 -> 13 -> Final

-----------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

cf , ef , sf , f , e

-----------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : cf

( 1,2 ):        1 -> 2
( 1,12 ):       1 -> 2 -> 12
( 1,13 ):       1 -> 2 -> 12 -> 8 -> 7 -> 13

Variable : ef

( 1,2 ):        1 -> 2
( 1,12 ):       1 -> 2 -> 12
( 1,8 ):        1 -> 2 -> 12 -> 8
( 1,7 ):        1 -> 2 -> 12 -> 8 -> 7
( 1,13 ):       1 -> 2 -> 12 -> 8 -> 7 -> 13

-----------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: sf
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: f
Type of Error: UN USED
```

Figure 6.9: Optimal trial 1 of Input Model of ECS

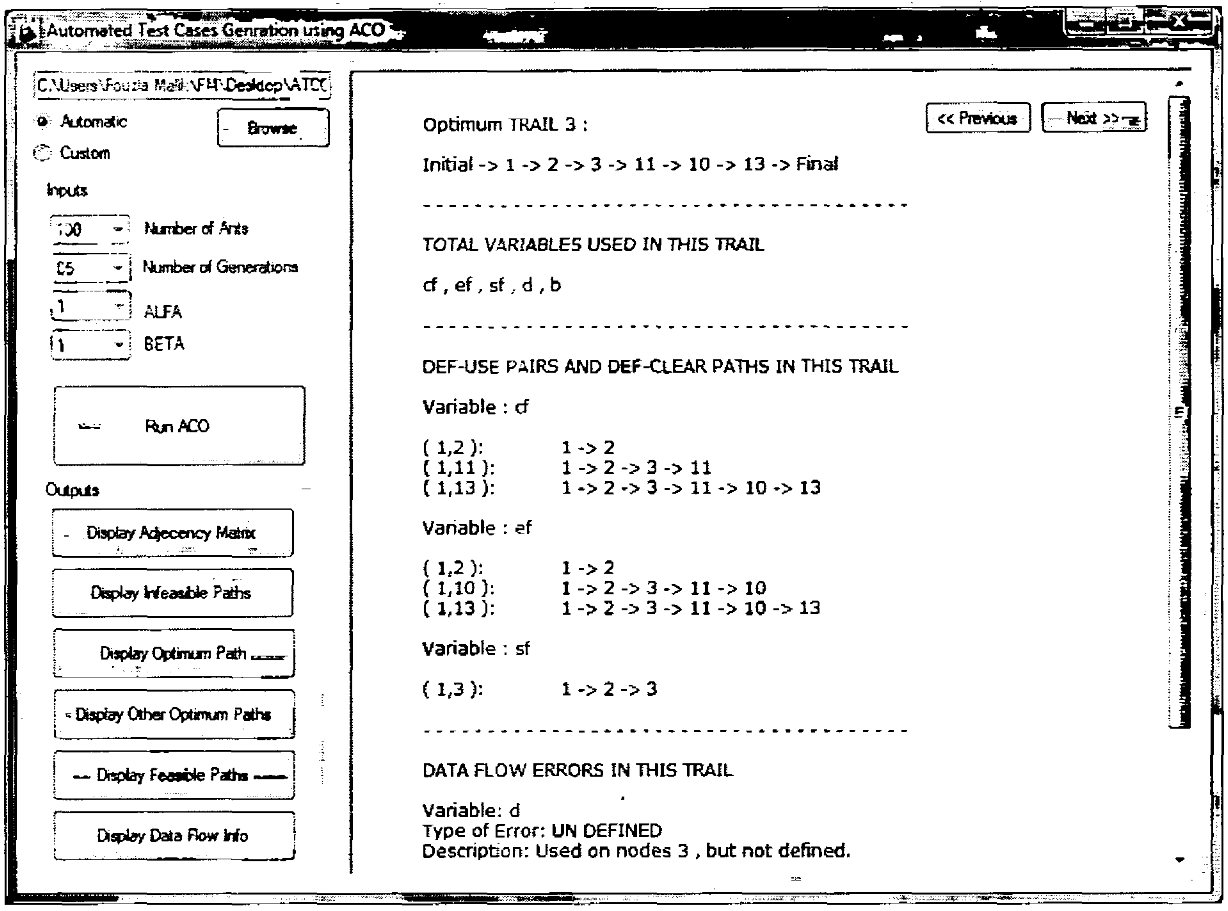
Optimum Trial 2 of Input Model of ECS



```
Optimum TRAIL 2 :

Initial -> 1 -> 9 -> 10 -> 13 -> Final

-------------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

cf , ef , sf , c , b

-------------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : cf

( 1,9 ):        1 -> 9
( 1,13 ):       1 -> 9 -> 10 -> 13

Variable : ef

( 1,10 ):       1 -> 9 -> 10
( 1,13 ):       1 -> 9 -> 10 -> 13

-------------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: sf
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: c
Type of Error: UN DEFINED
Description: Used on nodes 9 , but not defined.

Variable: b
```

Figure 6.10: Optimal trial 2 of Input Model of ECS

Optimum Trial 3 of Input Model of ECS

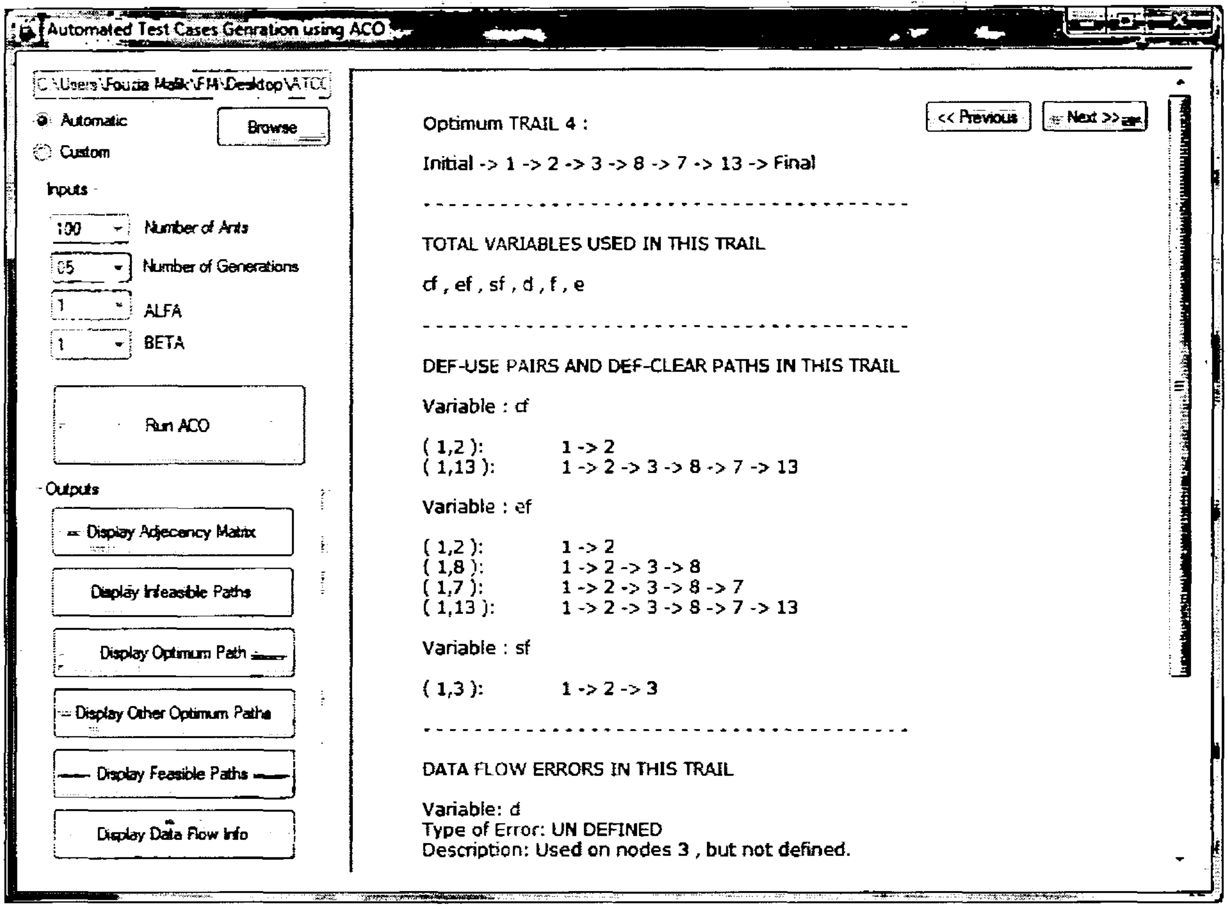


100 Number of Ants
05 Number of Generations
1 ALFA
1 BETA

```
Optimum TRAIL 3 :

Initial -> 1 -> 2 -> 3 -> 11 -> 10 -> 13 -> Final

-----------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

cf , ef , sf , d , b

-----------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : cf

( 1,2 ):        1 -> 2
( 1,11 ):       1 -> 2 -> 3 -> 11
( 1,13 ):       1 -> 2 -> 3 -> 11 -> 10 -> 13

Variable : ef

( 1,2 ):        1 -> 2
( 1,10 ):       1 -> 2 -> 3 -> 11 -> 10
( 1,13 ):       1 -> 2 -> 3 -> 11 -> 10 -> 13

Variable : sf

( 1,3 ):        1 -> 2 -> 3

-----------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: d
Type of Error: UN DEFINED
Description: Used on nodes 3 , but not defined.
```

Figure 6.11: Optimal trial 3 of Input Model of ECS

Optimum Trial 4 of Input Model of ECS

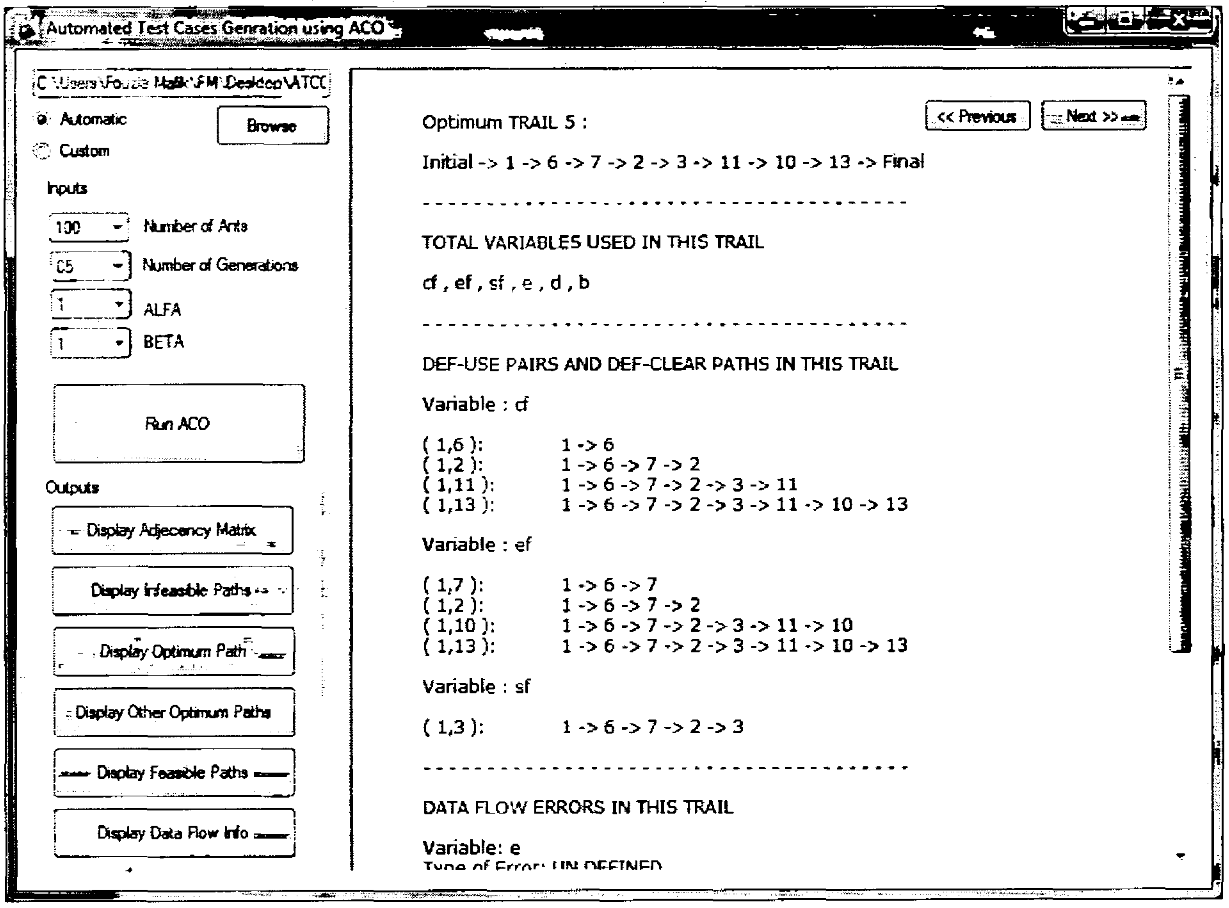


100 Number of Ants
05 Number of Generations
1 ALFA
1 BETA

```
Optimum TRAIL 4 :

Initial -> 1 -> 2 -> 3 -> 8 -> 7 -> 13 -> Final

-------------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

cf , ef , sf , d , f , e

-------------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : cf

( 1,2 ):        1 -> 2
( 1,13 ):       1 -> 2 -> 3 -> 8 -> 7 -> 13

Variable : ef

( 1,2 ):        1 -> 2
( 1,8 ):        1 -> 2 -> 3 -> 8
( 1,7 ):        1 -> 2 -> 3 -> 8 -> 7
( 1,13 ):       1 -> 2 -> 3 -> 8 -> 7 -> 13

Variable : sf

( 1,3 ):        1 -> 2 -> 3

-------------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: d
Type of Error: UN DEFINED
Description: Used on nodes 3 , but not defined.
```

Figure 6.12: Optimal trial 4 of Input Model of ECS

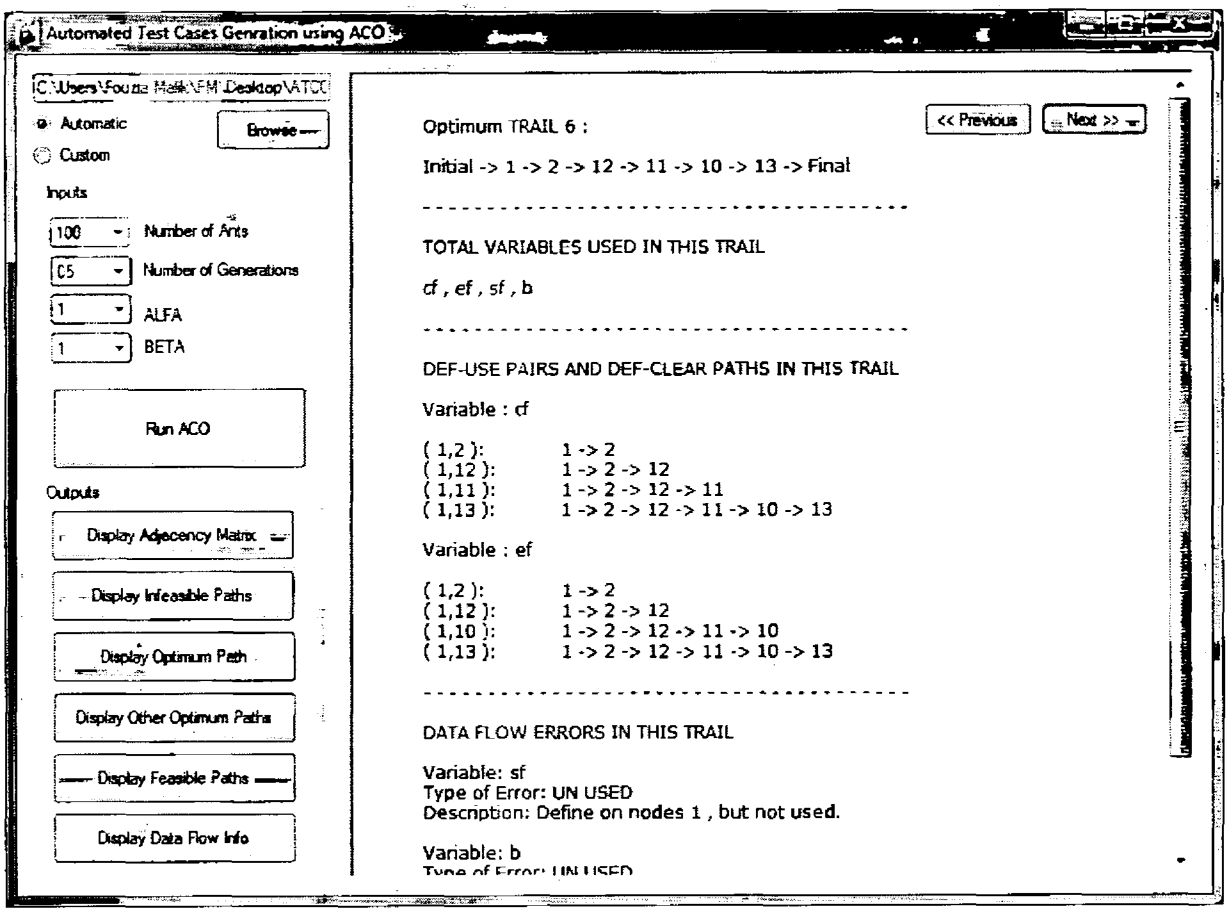
Optimum Trial 5 of Input Model of ECS



```
Optimum TRAIL 5 :

Initial -> 1 -> 6 -> 7 -> 2 -> 3 -> 11 -> 10 -> 13 -> Final

-----------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

cf , ef , sf , e , d , b

-----------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : cf

( 1,6 ):        1 -> 6
( 1,2 ):        1 -> 6 -> 7 -> 2
( 1,11 ):       1 -> 6 -> 7 -> 2 -> 3 -> 11
( 1,13 ):       1 -> 6 -> 7 -> 2 -> 3 -> 11 -> 10 -> 13

Variable : ef

( 1,7 ):        1 -> 6 -> 7
( 1,2 ):        1 -> 6 -> 7 -> 2
( 1,10 ):       1 -> 6 -> 7 -> 2 -> 3 -> 11 -> 10
( 1,13 ):       1 -> 6 -> 7 -> 2 -> 3 -> 11 -> 10 -> 13

Variable : sf

( 1,3 ):        1 -> 6 -> 7 -> 2 -> 3

-----------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: e
Type of Error: UN DEFINED
```

Figure 6.13: Optimal trial 5 of Input Model of ECS

Optimum Trial 6 of Input Model of ECS

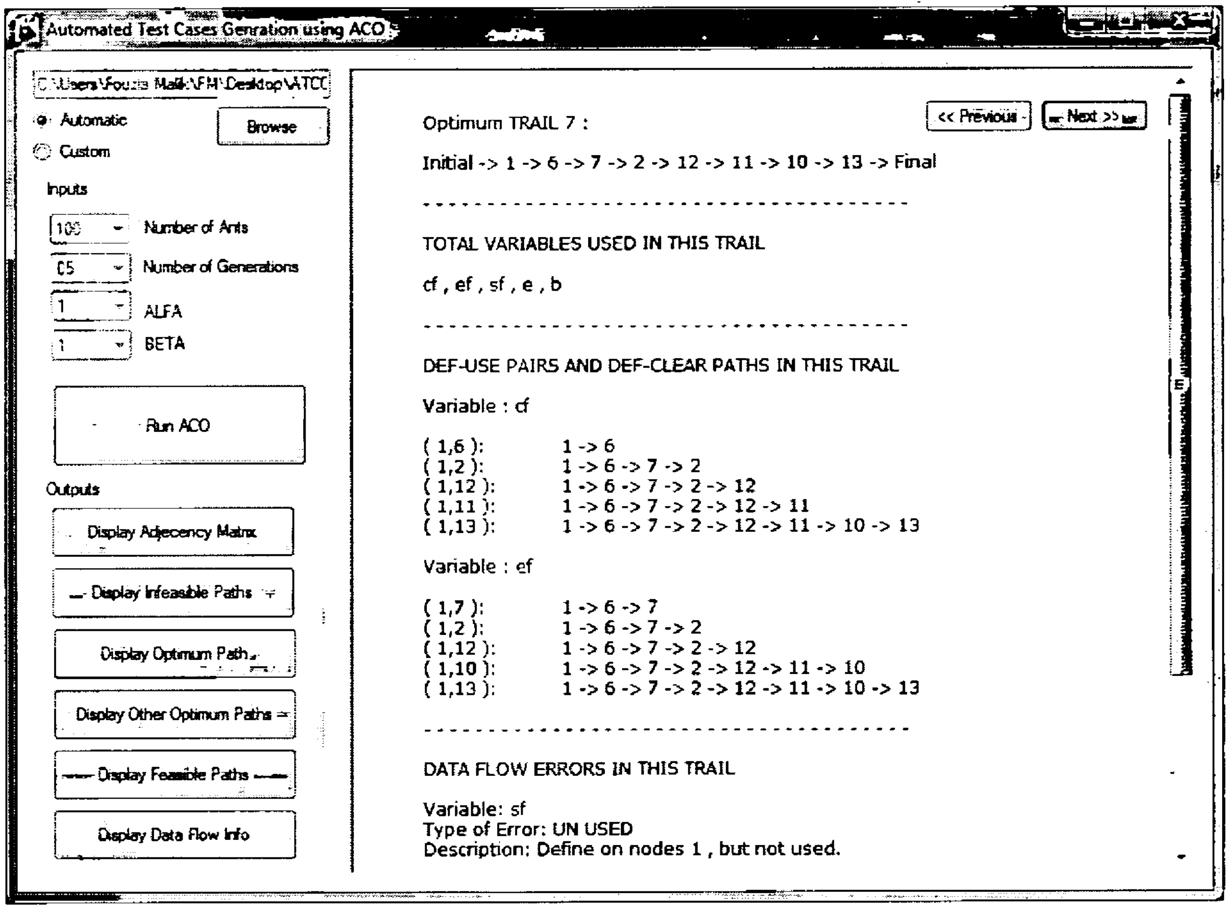


Number of Ants: 100
Number of Generations: C5
ALFA: 1
BETA: 1

```
Optimum TRAIL 6 :

Initial -> 1 -> 2 -> 12 -> 11 -> 10 -> 13 -> Final

-----------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

cf , ef , sf , b

-----------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : cf

( 1,2 ):        1 -> 2
( 1,12 ):       1 -> 2 -> 12
( 1,11 ):       1 -> 2 -> 12 -> 11
( 1,13 ):       1 -> 2 -> 12 -> 11 -> 10 -> 13

Variable : ef

( 1,2 ):        1 -> 2
( 1,12 ):       1 -> 2 -> 12
( 1,10 ):       1 -> 2 -> 12 -> 11 -> 10
( 1,13 ):       1 -> 2 -> 12 -> 11 -> 10 -> 13

-----------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: sf
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: b
Type of Error: UN USED
```

Figure 6.14: Optimal trial 6 of Input Model of ECS

Optimum Trial 7 of Input Model of ECS



100 Number of Ants
05 Number of Generations
1 ALFA
1 BETA

```
Optimum TRAIL 7 :

Initial -> 1 -> 6 -> 7 -> 2 -> 12 -> 11 -> 10 -> 13 -> Final

------------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

cf , ef , sf , e , b

------------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : cf

( 1,6 ):        1 -> 6
( 1,2 ):        1 -> 6 -> 7 -> 2
( 1,12 ):       1 -> 6 -> 7 -> 2 -> 12
( 1,11 ):       1 -> 6 -> 7 -> 2 -> 12 -> 11
( 1,13 ):       1 -> 6 -> 7 -> 2 -> 12 -> 11 -> 10 -> 13

Variable : ef

( 1,7 ):        1 -> 6 -> 7
( 1,2 ):        1 -> 6 -> 7 -> 2
( 1,12 ):       1 -> 6 -> 7 -> 2 -> 12
( 1,10 ):       1 -> 6 -> 7 -> 2 -> 12 -> 11 -> 10
( 1,13 ):       1 -> 6 -> 7 -> 2 -> 12 -> 11 -> 10 -> 13

------------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: sf
Type of Error: UN USED
Description: Define on nodes 1 , but not used.
```

Figure 6.15: Optimal trial 7 of Input Model of ECS

Chapter 7

CASE STUDY

7.1 Introduction

This chapter is devoted to presents the details of the case study to validate our approach. We choose example of an Elevator System to automatically analyze data flow and generate optimal number of test cases to fulfill coverage criteria by employing our approach. In this chapter, we describe the Elevator Control System by taking into account one UML State Machine diagram.

The rest of paper is structured as follows. Section 7.2 describes an overview of Elevator Control System and functional requirement of states. Section 7.3 demonstrates the state machine diagram.

7.2 Elevator Control System

This section consists of two subsections. The first subsection gives summary of the scope of the Elevator System. The second subsection describes the functional requirements of the system.

7.2.1 Scope of Elevator Control System

Elevator System manages the working and the behavior of elevator. The major part of the Elevator system is elevator which manages the working of elevator moving up, moving down, door open, door closed, resume door open, resume door closed and pick passenger from floors of buildings. The elevator is used in building having numerous floors ranging from 1 to n, where n is utmost number of floors within building.

The elevator has call buttons to move the elevator correspondence to each floor. On the base floor, there is one button to move up the elevator and at the middle floors except top, there are three buttons to move the elevator up and down and third one is to resume button that resume the moving elevator, either moving up or down. Top floor has only one button to move down the floor. When the elevator reaches the destination floor, door will also open. But user can also select another floor; door will be closed to move upward or move downward or presses the resume close button to move up the floor.

7.2.2 Functional Requirements

Idle/Door Closed: Elevator is at idle position when the door is closed. Elevator starts moving towards up, down the floor according to request of user. If current floor and ending floor is same than door will be open.

Start Moving Up: When the current floor is less than the ending floor, elevator starts moving down the floor. After reaching the ending floor, door will be open. But here user can also resume the door closed button and select another floor to move.

Start Moving Down: When the current floor is greater than the ending floor, elevator starts moving up the floor.

Idle/ Door Open Reached: When the elevator reaches the destination floor requested by the user then it will be moved to idle position.

Resume Moving Up: When the user reaches the requested floor except top, user will press the resume moving up button to move up the floor.

Resume Moving Down: When the user reaches the requested floor except base floor, user will press the resume moving down the floor button to move downward.

Floor Selected/ Door Closed: User can press this button to select a floor when elevator is at idle position or when user reaches the requested floor and still want to move to another floor than it presses this button for the selection of floor.

7.2.3 Details of Case Study

UML state machine is given as input to tool in the form of extensible markup language file (XML). XML is generated using UML modeling tool Enterprise architect. Tool extract states ids, transition ids and construct adjacency matrix. Adjacency matrix shows the feasible connection between states. If there is connection between states it shows 1 and if no connection exists between states than it indicates 0.For control flow information, we use this adjacency matrix to convert into control flow graph. This shows flow of information within states.

Data flow testing is performed using heuristic technique Ant Colony Optimization algorithm. In first generation, Ants start at initial state and move randomly to reach the final state. After completing its first tour ant updated pheromones value of paths that are traversed by all ants. Pheromone value is updated based on number of def-use pairs covered by a path divided by total number of nodes covered. Evaporation rate of pheromone value is set that larger path covering small number of def-use pairs and having low pheromone value has high evaporation rate than optimum path covering large number of def-use pairs and having high pheromone value has low evaporation rate. In second generation, ants don't move randomly. At each state ant check whether variable is defined or used and compute overall def-use pairs and def-use paths to traverse them. They start at initial state and calculate the probability, pheromones value. It also checks the states status which indicates the visited/unvisited status of states. For example if state is already visited than visited status is 0 and if status is unvisited than it is 1.Alpha and beta constant is set 1. Our coverage criterion is all du-paths. All uses of variables are computed by the approach. Repeated paths or redundant paths in each generation are eliminated. Optimum paths selected are based on providing coverage of maximum number of def-use pairs.

Adjacency Matrix

$$
\begin{array}{c|ccccccccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\hline
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
6 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
8 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
11 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
13 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
14 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
15 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

Fig. 7.1 Adjacency Matrix for Elevator State Machine

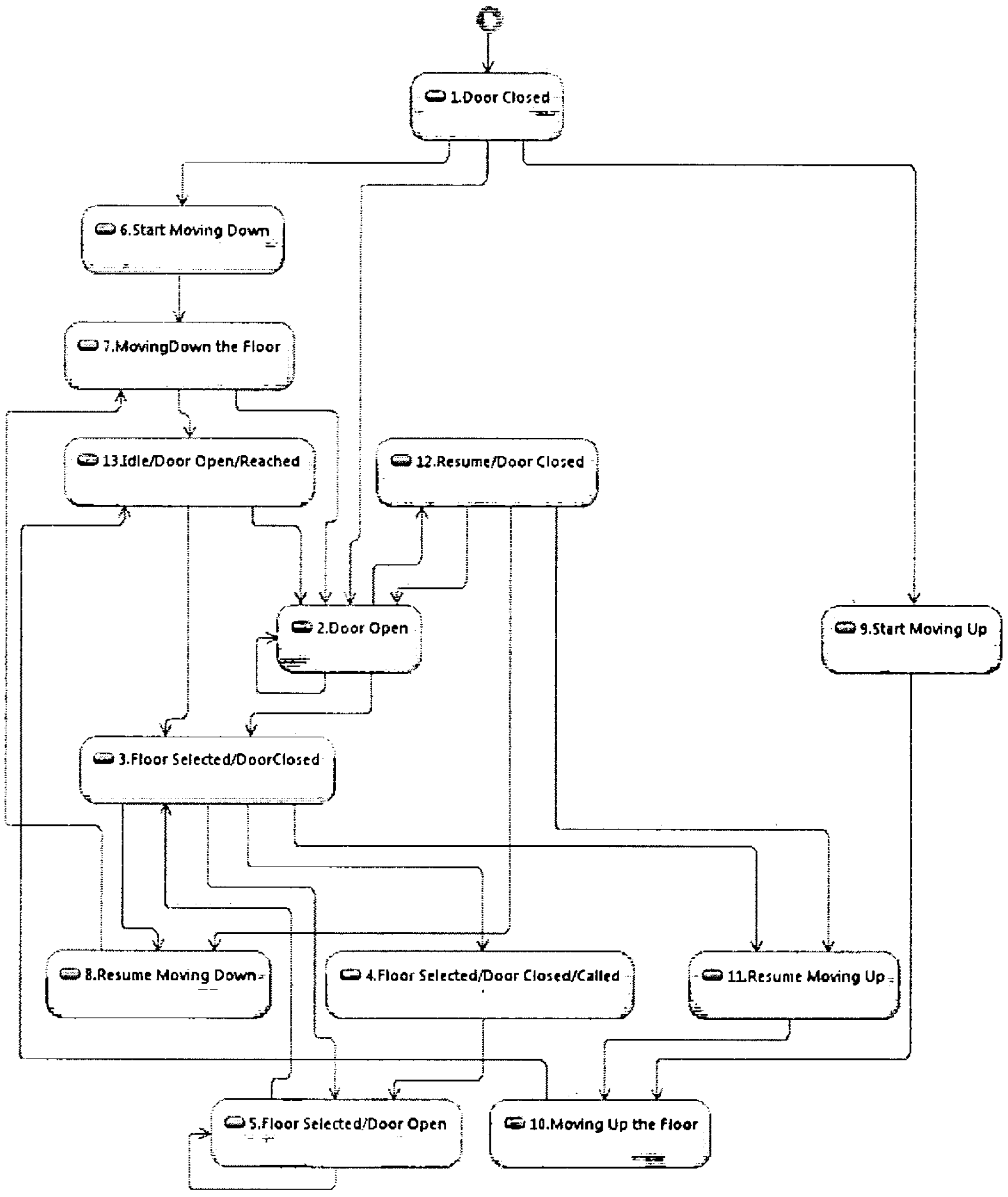


Fig. 7.2 State Machine Diagram of Elevator Control System

Consider the state machine diagram given in fig 7.2. Many ants are made to traverses the graph. In first generation ants take decision randomly because initially each state has zero probability and pheromones value. After completing first generation ants set pheromones value of all states that are made to traverse by them. In second generation, as ant approaches from initial state 1, ants calculate probability of all nodes that are directly connected with current/initial state and also check pheromones value. Ants traverse the states which have highest probability and pheromones value. For example at state 1 of state machine diagram in fig 7.2, there are three states that are directly connected with state 1; are state 2, 6 and 9. Ants calculate the probability of these three states. If these three states have equal probability than ants check visited status of these three states. If all are not visited yet than ant take decision randomly. If one of state is not visited from three states than ant select that state to traverse it. In state 2, 6, and 9, if state 6 has higher probability than state 2 and 9 than ant select 6 for next transition and current sequence becomes 1-6. Ants also store information of its current position along with position of nodes it made to traverse in a path. As mentioned above sequence 1-6, ant first store its current position that is at state 1.when ant select next transition to move it also maintain record of position of node and path.

During its tour, ants also analyze data flow within states. When ants select nodes to move, it checks whether the state contains the definition-use pair. It counts the definition-use pair exists within state and also checks the definition-use paths covered in a test path. Definition-use path is path from variable definition to node where it is used. And count total number of def-use pair along its paths. All paths are selected by from where every variable definition occurs to every use of that definition. As first tours of ants are random, suppose ant traverse the paths are;

1->6->7->13

1->2->12->11->10->13

1->9->10->13

The probability of optimal paths in each generation is high calculated using the formula in (1).

$$P = \{(\tau ij)^{\alpha} * (\eta ij)^{\beta}\} / \Sigma (\tau ij)^{\alpha} * (\eta ij)^{\beta} \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

- P = Probability
- τ = Pheromones Value
- η = Desirability Factor
- Alpha & Beta are constants

Ants after completing their tour, set pheromones value of paths that it traverse. Every solution searched by ACO is evaluated against fitness function given in (2) that we have tailored for state based data flow testing problem.

$$PV = \frac{DU}{TN} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2)$$

- PV = Pheromones Value
- DU = Total Definition-Use Pairs covered by a path
- TN = Total Number of Nodes covered by a path.

The fitness value produced using fitness function indicates the appropriateness of optimal solution.

Consider the first test path 1->6->7->13. Ants identify the def-use pair and definition-use path from variable definition to its use without definition of variable again in a same path. At state 1, there are 2 variables that are defined while used at state 6, 7 & 13. So 3 def-use pairs and def-clear paths exists in this path.

Cf (1, 6) = 1->6

Cf (1, 13) = 1->6->7->13

Ef (1, 7) = 1->6->7

So pheromone value of test sequence 1 is 3/4 = 0.75

This value is set on each state (6, 7, and 13) that is covered by ants in path 1.

For path 2;

1->2->12->11->10->13

The number of Def-use pairs and def-use paths covered in this test path are;

Cf (1, 2) = 1->2

Cf (1, 13) = 1->2->12->11->10->13

Ef (1, 2) = 1->2

Ef (1, 10) = 1->2->12->11->10

Ef (1, 13) = 1->2->12->11->10->13

So pheromone value of path 2 is 5/6 = 0.833

Consider the third test path1->9->10->13. Def-use pairs and def-use paths covered in this path are;

Cf (1, 9) = 1->9

Cf (1, 13) = 1->9->10->13

Ef (1, 10) = 1->9->10

Ef (1, 13) = 1->9->10->13

In this path, 4 def-use pairs and def-use paths exists. Pheromones value set by ants on 3 test case is 4/4 = 1.0

As pheromone value of test case 3 is higher than test case 1 and 2, so it also has higher probabilty than test case 1 and 2 .

For example, at state 1 ants checks the pheromone value and P-factor of state 2,6,9 that are one transition apart from state 1. State 9 have highh P-factor also having greater pheromone value than state 2 and 6. At this point ants checks the visited status of both states, if both states are not visited yet then ants decide randomly to move to next state. Suppose ants select state 9 due to its higher p factor. At state 9, ants check again the desirability factor and pheromone value of states that are directly connected from this state. State 10 is the only state that is directly connected from state 9. So ants moves to state 10. At state 10, it is only connected with with state 13. Ants check its visited status and mone to this state, a final state of model.And current sequence becomes 1-9-10-13. After completing each tour by ants, pheromone value is updated on traverse nodes of all paths of graph.

All definition-use pairs are identified by ants during their traversal of graph. All Definition-use paths of each definition-use pairs is calculated. Definition-use path is path that start at state where variable definition occurs to place where that variable is used. We have identified all the def-use paths of each definition-use pair. Each definition-use pair has several definition -use paths. One definition-use pairs may have one or more than one def-use paths as mentioned in table 7.1. For example, variable cf has deinition-use pair [1, 13] which has six def-use paths [1->2->3->8->7->13], [1->9->10->13], [1->6->7->2->3->11->10->13], [1->6->7->13], [1->6->7->2->12->11->10->13], and [1->2->12->8->7->13]. After the identification of def-use paths, test cases are generated to cover these definition-use paths. Each def-use path is examined and redundant definition-use paths are removed by tool. For example, as in table 7.1 the def-use paths identified by tool are [1->6->7->13], [1->6], [1->6->7->2], [1->6->7->2->3]. Aforementioned 4 definition-use paths, a single test case can cover these def-clear paths. That is [1 -> 6 -> 7 -> 2 -> 12 -> 11 -> 10 -> 13] cover the [1->6->7->13], [1->6], [1->6->7->2], [1->6->7->2->3]. All these def-clear paths are included in this test case.

All paths that are traversed by ants, p factor is calculated for each path. Ants decision to move to next state is based on pheromone value and probaility. These values of pheromones help other ants as traces to select a path to destination. To avoid the local optima, evaporation rate is considered. Pheromone value is evaporated based onformula given in 3.

$$\tau ij = (1-r)\, \tau ij + \Delta\tau ij \quad \text{...} (3)$$

r= Rate of evaporation of pheromones

$\Delta\tau ij$= Total amount of pheromones set down by ants when it traverse from edge i to j.

Using formula 3, pheromones of each path evaporates. Path covering large number of definition-use pairs have low evaporation rate than the path covering low number of definition-use pair in which pheromone value evaporate quickly.

<u>Overall def-use pairs/def-clear paths</u>

No of Variables used	Variable	DU-pair	Def-Clear Path
1.	Cf	1,6	1->6
2.	Cf	1,9	1->9
3.	Cf	1,2	1->2 1->6->7->2
4.	Cf	1,12	1->2->12
5.	Cf	1,13	1->2->3->8->7->13 1->9->10->13 1->6->7->2->3->11->10->13 1->6->7->13 1->6->7->2->12->11->10->13 1->2->12->8->7->13
6.	Cf	1,11	1->2->3->11
7.	Ef	1,10	1->9->10 1->6->7->2->3->11->10 1->6->7->2->12->11->10
8.	Ef	1,2	1->2 1->6->7->2
9.	Ef	1,12	1->2->12
10.	Ef	1,8	1->2->3->8
11.	Ef	1,13	1->2->3->8->7->13
12.	Ef	1,7	1->6->7 1->2->3->8->7 1->2->12->8->7
13.	Sf	1,3	1->2->3 1->6->7->2->3

Table 7.1 Def-use pairs of the variables used within State Machine

In the same way, ants made to traverse the graph and update the pheromones value.If ants at some point has state that are directly connected to current state has same probability and also has same desirabilty factor than ants select randomly. If ants at some point have state that are directly connected to current states has same probability than ants move to state that is unvisited. Redundant number of test cases are avoided by resticting the ants to not move to state that is already visited. In each tour, ants visit one state at least one time. So ants will explore different paths by traversing the graph.

7.2.3.1 Infeasible paths

Paths that are considered infeasible are;

- Ants stuck in cycle due to consideration of states visited status because ants don't traverse the node that is already traversed.
- Paths that don't reach to its destination.
- Paths in which there is no connection between states.
- State having dead end.

7.2.3.2 Mutation Testing

To determine the error detection rate of state machine model, we introduce faults into states of model. Mutation testing is widely used yielding realistic results and is frequently used during testing research [9].

Tool detects the following data flow errors as well as the mutated faults.

- Variable is defined in state but not used in a given diagram.
- Variable is defined in one state and also defined in another state on a same path before its use (redefinition of variable in a state on same path is a killing state).
- Variable used but not defined.
- Multiple usage of variable but not defined.
- Multiple definitions of variable but not used.
- Variable used before it is defined.

In state machine diagram of fig 7.2, 34 errors are seeded within states of model. Tool detects seeded errors against each path. Mutation score of all models are listed in table 7.3.

Model#	Model	States Coverage	Def-Use Paths	Def- Use Path Coverage	Mutation Score (%)
1	ECS	84.6	43	95	95.2
2	TS	100	48	100	100
3	POS	100	13	100	100
4	HMS	100	8	100	100
5	ATM	100	7	100	100
6	LMS	87.5	7	80	83.7
7	SES	88.8	5	83.3	90
8	CC	100	4	100	100
9	DM	100	10	100	100

Table 7.2 Showing mutation Score of All Models

Fig 7.2 illustrates the total du-paths within ECS model and mutation score of ECS model. There are 43 def-use pairs within ECS model, 41 of which are covered by tool. Mutation score of ECS model is 95.2%. As listed above in table 7.2. TS model have all definition-use paths coverage having high mutation score. There is direct relationship exists between rate of mutation and number of def-clear paths. Models that have provide coverage of all def-clear paths have high mutation score. Some of the models have same mutation score due to same coverage of def-clear paths. The analysis advocates that traversing certain def-clear paths make certain faults to be detected.

Chapter 8

EVALUATION

8.1 Introduction

This chapter is dedicated to clarify and assess experiment results which are performed to authenticate our approach for data flow testing. Section 8.2 describes the results of our approach on the model of the Elevator System that is elaborated in chapter 7.The validation results are presented in section 8.3. Comparison of our approach with existing state based data flow testing approaches and tools is discussed in section 8.4. Overall benefits and limitations are given in section 8.5.

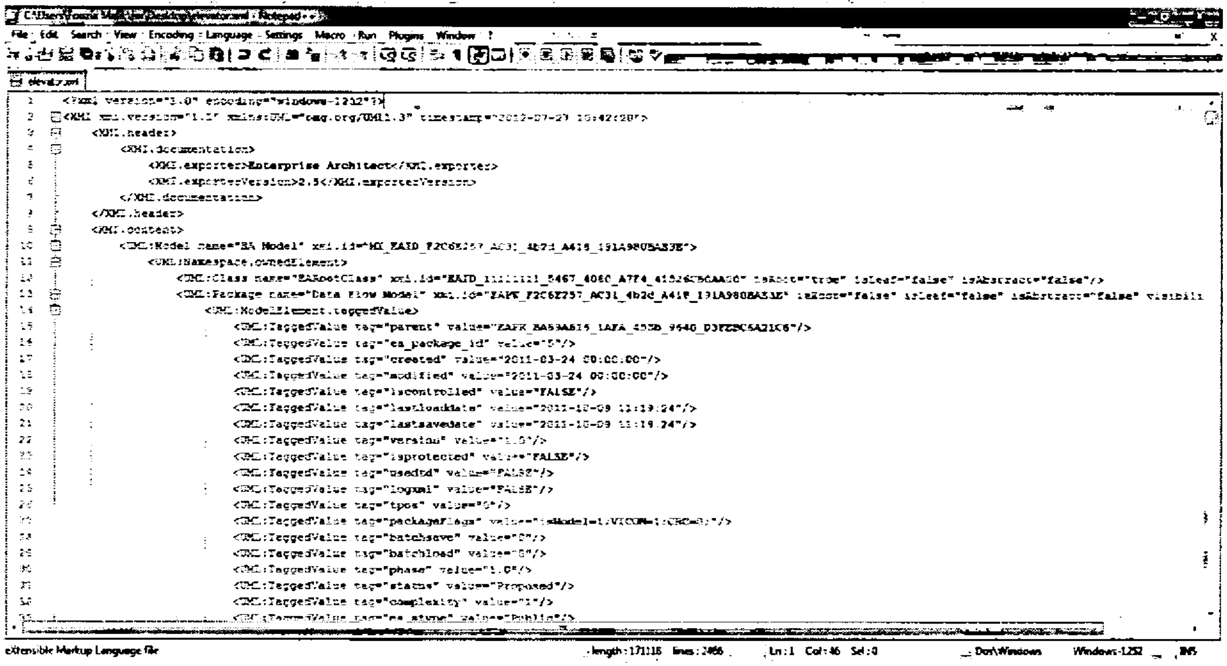
8.2 Elevator Control System

In this part, we explain the experimental setting and results are discussed by applying our approach to state machine model of Elevator Control System.

8.2.1 Experimental Setting of Tool

The experiment steps are summarized below.

1. In the first step, we generate XML of state machine models using UML modeling tool Enterprise Architect. The state machine models of 9 different software systems are used to generate XML. The XML format of all models can be found in Appendix A. For Elevator Control System, XML is stored in elevator.xml file. We store the 9 XML of 9 different software system in 9 different files; one file contains the XML of one state machine model. This XML is given as input to our data flow testing tool. Figure 8.1 depicts the XML file format of ECS model.



```
<?xml version="1.0" encoding="windows-1252"?>
<XMI xmi.version="1.1" xmlns:UML="omg.org/UML1.3" timestamp="2012-07-27 15:42:20">
    <XMI.header>
        <XMI.documentation>
            <XMI.exporter>Enterprise Architect</XMI.exporter>
            <XMI.exporterVersion>2.5</XMI.exporterVersion>
        </XMI.documentation>
    </XMI.header>
    <XMI.content>
        <UML:Model name="EA Model" xmi.id="MX_EAID_F2C6E257_AC31_4b2d_A418_191A980BA33E">
            <UML:Namespace.ownedElement>
                <UML:Class name="EARootClass" xmi.id="EAID_11111111_5467_4080_A7F4_41526CBCAA00" isRoot="true" isLeaf="false" isAbstract="false"/>
                <UML:Package name="Data Flow Model" xmi.id="EAPK_F2C6E257_AC31_4b2d_A418_191A980BA33E" isRoot="false" isLeaf="false" isAbstract="false" visibili
                    <UML:ModelElement.taggedValue>
                        <UML:TaggedValue tag="parent" value="EAPK_BA59A615_1AFA_455b_9640_D3FEBC5A21C6"/>
                        <UML:TaggedValue tag="ea_package_id" value="5"/>
                        <UML:TaggedValue tag="created" value="2011-03-24 00:00:00"/>
                        <UML:TaggedValue tag="modified" value="2011-03-24 00:00:00"/>
                        <UML:TaggedValue tag="iscontrolled" value="FALSE"/>
                        <UML:TaggedValue tag="lastloaddate" value="2011-12-09 11:19:24"/>
                        <UML:TaggedValue tag="lastsavedate" value="2011-12-09 11:19:24"/>
                        <UML:TaggedValue tag="version" value="1.0"/>
                        <UML:TaggedValue tag="isprotected" value="FALSE"/>
                        <UML:TaggedValue tag="usedtd" value="FALSE"/>
                        <UML:TaggedValue tag="logxml" value="FALSE"/>
                        <UML:TaggedValue tag="tpos" value="0"/>
                        <UML:TaggedValue tag="packageFlags" value="isModel=1;VICON=1;CRC=0;"/>
                        <UML:TaggedValue tag="batchsave" value="0"/>
                        <UML:TaggedValue tag="batchload" value="0"/>
                        <UML:TaggedValue tag="phase" value="1.0"/>
                        <UML:TaggedValue tag="status" value="Proposed"/>
                        <UML:TaggedValue tag="complexity" value="1"/>
                        <UML:TaggedValue tag="ea_stype" value="Public"/>
```

eXtensible Markup Language file   length : 171116   lines : 2466   Ln : 1   Col : 46   Sel : 0   Dos\Windows   Windows-1252   INS

Figure 8.1 XML file format of ECS

2. During the second step, control flow graph is created from adjacency matrix of input model, which shows the total number of states and feasible connections between them.
3. Automated analysis of data flow and minimal paths are generated to fulfill the required all def-use path coverage criteria.
4. Mutation testing is performed to analyze the effectiveness of our approach.

8.2.2 Results and Discussion

Table 8.1 shows the results generated by data flow testing tool. According to the result, from the total of 9 state machine models, there are 3 models that don't provide complete coverage while all other models provide 100% definition-use path coverage.

As we have seen in table 8.1, shows that model have high coverage of def-use paths have high mutation score. Library management (LMS) has lowest def-use path coverage having less mutation score than other models. It shows that models that exercise more def-use paths have detect more data flow faults.

No. of Models	Model	Data Flow Criteria	DU-Paths Coverage	States Coverage	Mutation Score (%)
1	ECS	All du-paths	95%	84.6%	95.2%
2	TS	All du-paths	100%	100%	100%
3	ATM	All du-paths	100%	100%	100%
4	SES	All du-paths	83.3%	88.8%	90%
5	CC	All du-paths	100%	100%	100%
6	POS	All du-paths	100%	100%	100%
7	LMS	All du-paths	80%	87.5%	83.7%
8	HMS	All du-paths	100%	100%	100%
9	DM	All du-paths	100%	100%	100%

Table 8.1 DU-paths Coverage &Mutation Scores of all Models

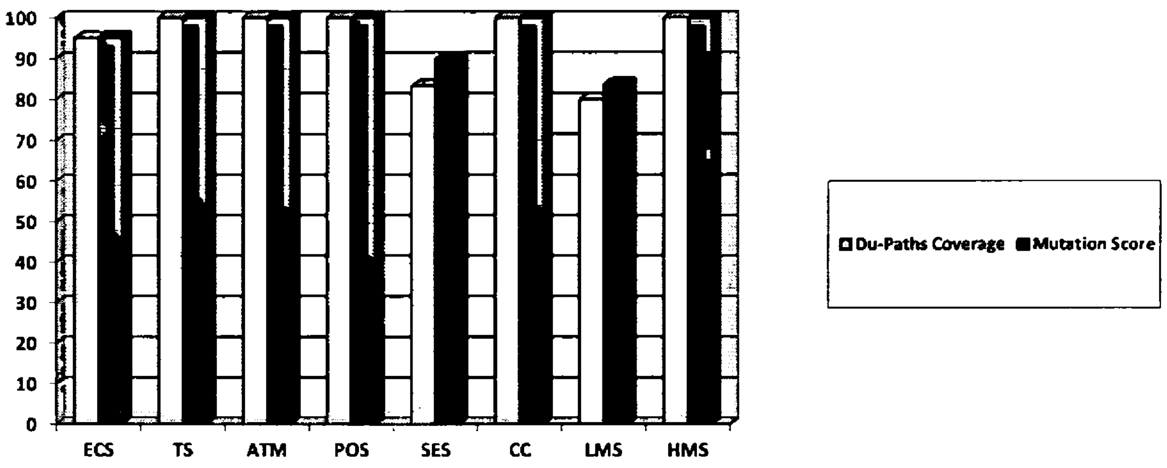


Figure 8.2 def-use Paths vs. Mutation Score

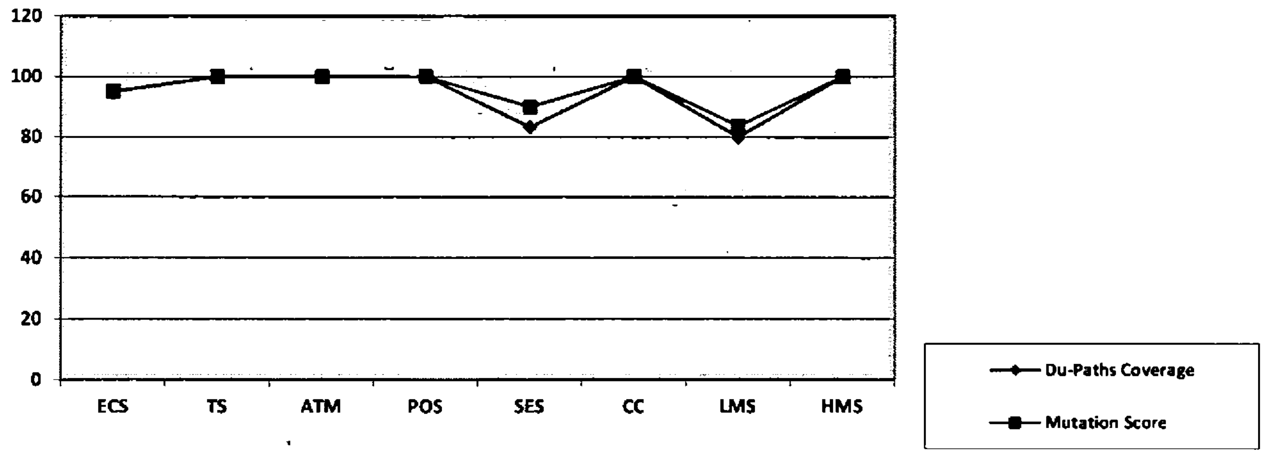


Figure 8.3 def-use Paths vs. Mutation Score

We also evaluate our approach based on *general coverage* including states coverage provided by existing techniques and our proposed technique. These techniques are also evaluated against *fault wise coverage* provided by them. Different categorizes of software faults are identified from literature [51, 13, 8] and all techniques are evaluated against them. Detection of these software faults is necessary because the presence of software errors in software results in disaster [51]. As our coverage criteria are all definition-use paths. The variable is considered as define when variable is assigned a value or in memory value is stored. Node/State containing the definition of variable is known as def or definition node. The variable is considered as use when that value is fetched from memory or value that is defined is used. Node contacting the use of variable that is defined before is known as used node. Variable from its point of definition to its use is known as def-use pair. Path between definitions of variable to its possible use is known as definition-use path. According to coverage criteria, we will include all paths from variable definition to the point it uses. We include or select overall definition-use pairs and select all paths of every definition-use pair. The possible combination of definition (d), killed (k) and used (u) that are valid or acceptable are;

- **Define and Used (du)**

Variable is defined before it's used within state. This is correct combination of variable definition and its possible use.

Some of the standard definition and usage faults/anomalies identified from literature are;

- ***Define Without Use (d~) [51,13,8]***

  This is anomalous because variable is defined within state of model but it is not used in any state. Since variable is defined, it is not used lead to programming error.
- ***Double Define (dd) [51,8]***

  Variable that is defined in a state and is redefined which is not used is invalid, suspicious as well as programming error lead to serious disaster.
- ***Use without Define (~u) [51,13,8]***

  This is a serious problem that state uses a variable which is not defined in any path of model. Usage of undefined variable is a software fault.
- ***Double Use (uu) [8]***

  This is also an error when variable is used more than one time but is not defined in any state within model.
- ***Defined without used in Scope (dk)*** **[8]**

  This is a programming error that variable is defined but not used in path and is killing in the same path.
- ***Data is used beyond the Scope (ku) [8]***

  That is a serious defect that after variable is kill it is used on the same path.
- ***Variable is defined more than once before use [13]***

  This is also Programming error because as we are considering def-use paths in which is path from variable definition to its usage without any definition of variable again in same path.

These are the possible state based standard faults that can occur in states. And we evaluate existing approaches and proposed approach against them. We also mutate these possible state based faults in our input state machine models and our proposed approach detects these mutated faults. The faults detected by the existing approaches and our proposed approach are given below in a table 8.2. Faults that are identified from literature are mutated in input telephone system model, and mutated state machine model is shown in fig 8.4.

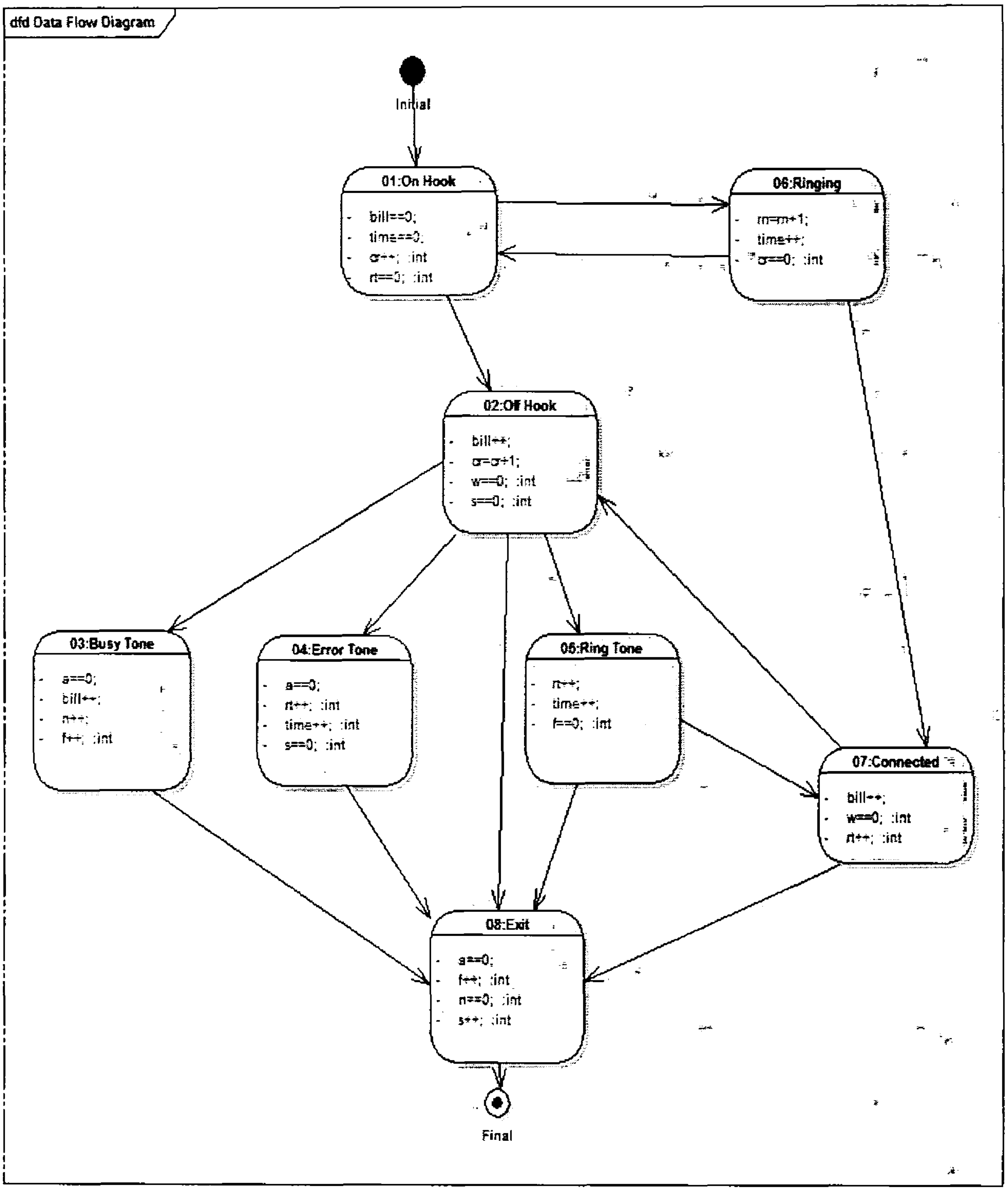


Fig 8.4 Mutated Telephone System Model

Errors Type	*Number of Seeded Faults*
~d	15
u~	10
Dd	6
Uu	4
Used Before Define	5
Dk	1
Ku	1
Average	42

Table 8.2 Types and Number of Errors Seeded

Model	Number of Faults Seeded	Mutation Score	Number of Test Case
Telephone System Model	42	100	6
Telephone System Model	38	90.47%	5

Table 8.3 Mutation Score and Number of test Cases Needed

We mutated faults within telephone system model overall containing 42 faults. We have seeded faults within states of input model, as we are only focusing on state coverage. Test cases are generated to cover all these states. Our tool automatically generates test cases to detect these seeded faults and definition-use paths were used as a coverage criterion. In telephone model, a number of possible paths are generated but due to generation of non-redundant test cases we don't allow ant to revisit same node twice that's why there are total 10 paths within input model. As aforementioned we have seeded 42 faults, out tool generate optimal number of test paths and provide efficient detection of data flow faults with 6 numbers of test cases that provide 100% mutation score. With 5 numbers of test cases, our tool provides 90.47% mutation score the fault missing are the 2 used before defined and 2 of faults variable used but not defined in any path of model. With minimal number of 6 test cases our tool is able in detecting overall seeded 42 faults.

Fault-Wise Coverage:

Existing Paper Author Names	Algorithm Used	Data flow Criteria Used	Faults Overall Coverage	Define, Not Used (d~) [51,13,8]	Used, Not Define (~u) [51,13,8]	Double Define (dd) [51,8]	Double Use (uu) [8]	Variable is Defined More than Once Before Use [8]	Defined without use in scope (dk) [8]	Data is used beyond the scope (ku) [8]	Used Before Defined
L.C.Briand	------	All Du-Paths	88%,96%	----	----	-----	-----	-------	-----	-----	-----
31	------	All Du-Paths	96%	-----	-----	-----	----	------	----	-----	-----
S.Andreous	------	All-uses All Du-Paths (Code Based)	100%	Yes	Yes	Yes	Yes	------	Yes	Yes	----
H.K Dubeu	-------	All Du-Paths (Code Based)	-------	Yes	Yes	-------	-------	Yes	-------	-------	-----
L.C.Briand	------	-------	-------	Yes	Yes	Yes	-------	-------	-------	-------	-------
Proposed Approach	ACO	All Du-Paths	100%	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 8.4 Fault-wise coverage provided by Existing Techniques and Proposed Technique (------ = No)

As aforementioned in table 8.4, two of the model based approaches [9] [31] perform mutation testing and detect data flow faults but don't mention categorizes of faults that they identified or detected by their tool. None of the other model based approach performs mutation testing. One of the two code based approach [6] detect almost all faults except variable is defined more than one time before its use. Approach in [13] detect only three type of faults; variable is define but not used, Variable is used without defining, and variable is defined more than one time before it use. But or proposed approach detect all categorizes of software faults as well as seeded faults.

General Coverage Provided by Code Based Techniques:

Existing Paper no	Author Name	Algorithm Used	Coverage Criteria Used	Program Coverage	Mutation Testing Performed
[28]	Ahmed S. Ghiduk	GA	All-Uses	100%	No
[3]	Sanjay Singla	PSO	All-Uses	100%	No
[32]	Moheb R. Girgis	GA	All-Uses	100%(when no infeasible paths exists)	No
[6]	Andreas S. Andreou	GA	All Du-Paths	100%	No
[8]	Jun Hou	-------	All-Uses All Du-Paths	100%	Yes
[13]	Harsh Kumar Dubey	--------	All Du-Paths	-------	No

Table 8.5 General coverage provided by Existing Code-Based Techniques (------ = No)

General Coverage Provided by Model Based Techniques:

Existing Paper no	Author Name	Algorithm Used	Coverage Criteria Used	States Coverage	Mutation Testing performed
[33]	Huaizhong LI	ACO	All States Coverage	100%	No
[9]	L.C. Briand	-------	All Du-Paths	--------	Yes
[31]	L.C. Briand	-------	All Du-Paths	--------	Yes
[29]	Chartchai Doungsa-ard	GA	All-Transitions	100% coverage if system don't contain final state	No
[27]	Praveen Ranjan Srivastava	ACO	All-Transitions	100%	No
[10]	Tabinda	------	All Du-Paths	-------	No
[11]	Praveen Ranjan Srivastava	ACO	Criticality of states	100%	No
Proposed Approach	Fozia, Atif Aftab Ahmed Jilani	ACO	All Du-Paths	100%	Yes

Table 8.6 General coverage provided by Model-Based Existing Techniques and Proposed Technique (------ = No)

No. of Pairs	*Variable Used*	*Definition-Use Pairs*	*All Definition Use Paths*
1	Cf	(1.3.2.2)	1-2 1-7-2 1-9-10-2
2	Cf	(1.3,5.2)	1-2-3-5 1-2-3-4-5
3	Cf	(1.3,6.3)	1-6
4	Cf	(1.3,9.3)	1-9
5	floorNo	(1.3,7.1)	1-6-7
6	floorNo	(1.3,10.1)	1-9-10 1-2-12-10 1-2-3-11-10
7	floorNo	(7.1, 7.1)	7-7
8	floorNo	(7.1,10.1)	7-2-12-11-10 7-2-3-11-10
9	Ef	(1.4,3.4)	1-2-3
10	Ef	(1.4,7.1)	1-6-7
11	Ef	(1.4,10.1)	1-9-10
12	Ef	(7.1,3.4)	7-2-3
13	Ef	(7.1,7.1)	7-7
14	Ef	(7.1,10.1)	7-2-12-11-10 7-2-3-11-10
15	Sf	(1.5,3.1)	1-2-3
16	Sf	(1.5,3.2)	1-2-3

Table 8.7 Total definition-Use Pairs & All definition-use Paths in paper [10]

When comparing with approach in [10], they validate their approach on elevator state machine model and only identify the definition-use pair. In elevator system, there are 16 definition-use pairs and 24 Definition-use paths. We generate test cases to satisfy these definition-use paths. Existing technique needs 9 test cases to satisfy all these def-use paths and provides 100% definition-use paths as well as states coverage. While our proposed approach identify all the definition-use paths within model and remove the redundant ones. By using ACO, optimal number of test cases is generated to satisfy these def-use paths. Our proposed approach needs 7 test cases to satisfy all these def-use paths coverage criteria to provide 100 % definition use paths coverage and states coverage.

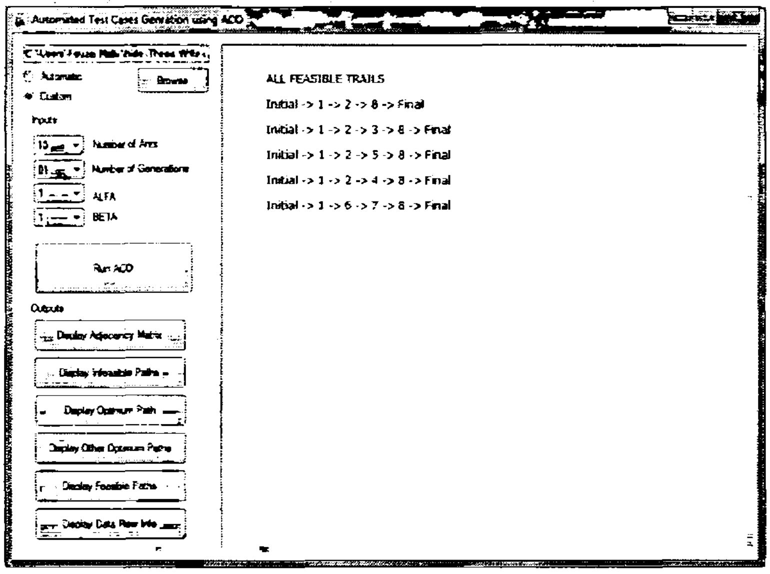
Ranjan [11] used Ant colony optimization for generation of test cases in a state based system. But they focused on criticality of the states generating optimized test sequences and don't consider data flow coverage. By using the results achieved by the application of technique in [11] and the proposed technique, we make a comparative study. We choose this approach for comparison as this work include ACO in model based testing for test case generation purpose. As existing approach don't analyze data flow so we compare the coverage provided by technique in [11] and our proposed approach. Both techniques use Telephone system model. Model contains 8 states and 15 transitions. The number of test cases required by existing technique providing complete coverage is 6;

1) 0 ->2-> 3-> 7
2) 0->1-> 6 ->2->3-> 7
3) 0-> 1 ->1-> 0-> 2 -> 5 -> 7
4) 0 -> 2 -> 4 -> 7
5) 0-> 2-> 7
6) 0-> 1 --> 6-> 7

And the proposed technique requires only 5 test cases to provide complete coverage.

1) 1 -> 2 -> 8
2) 1 -> 2 -> 3 -> 8
3) 1 -> 2 -> 5 -> 8
4) 1 -> 2 -> 4 -> 8
5) 1 -> 6 -> 7 -> 8

Existing approach consider states 3, 4, 5 and 6 as critical states and cost factor but our approach consider all definition-use paths as coverage criteria and detect data flow errors. According to existing approach our approach provides full state coverage including critical states in 5 test cases that they provide in 6 test cases. Secondly redundant numbers of states are covered in providing coverage that our approach doesn't cover still providing complete coverage. As exiting approach consider cost factor, under limit of cost 20, our approach provide coverage with 5 test cases but existing provide it with 6.



```
ALL FEASIBLE TRAILS
Initial -> 1 -> 2 -> 8 -> Final
Initial -> 1 -> 2 -> 3 -> 8 -> Final
Initial -> 1 -> 2 -> 5 -> 8 -> Final
Initial -> 1 -> 2 -> 4 -> 8 -> Final
Initial -> 1 -> 6 -> 7 -> 8 -> Final
```

Fig 8.5 Automatically Generated Test Cases of Telephone System Model

When considering data flow coverage, existing aforementioned approach don't analyzes the flow of data just focused on control flow while our proposed approach analyzes and detects the data flow errors. Total number of definition-use pairs in model is 9 and there are 16 definition-use paths as mentioned in table 8.7. Our approach with a single generation of ants detects 9 def-use pairs and 11 definition use paths and efficiently detects all the data flow errors. The 20 faults are mutated within states of model that are detected by our approach with a single generation of ants. And also identify the all definition-use pairs. In our proposed approach, total 2 generations are required to cover all definition-use paths. In 2 generation, it provides all def-use pairs, definition-use paths, and all states coverage.

It is quite obvious that our proposed approach not only ensures a better state coverage but also ensures that it is intelligent enough in detection of data flow faults.

H Li [33] have used ACO for state based testing and automatically generate test cases. This approach focused on all states coverage don't focus on analysis of data flow within model. If comparing test cases generated by technique in [33] and our proposed approach, than existing approach provides state coverage with redundant number of test cases. Ranjan [27] also used ACO focusing on all transition coverage don't perform analysis of flow of data.

Comparing our proposed approach with approach in [10], both the techniques use same elevator control system model. But existing approach [10] only identifies def-use pairs but don't generate test cases to provide the coverage. Secondly mutation testing is also not performed by this approach that our proposed approach does.

Comparing it with technique [9] perform data flow analysis, identifies def-use pairs but their approach result in incomplete def-use paths coverage. Due to some faults remained undetected and also manually identify infeasible paths. They validate their approach on two state machine models. One of cruise control model which our approach also use for validation. Approach in [9] provides 96% fault detection rate and cover 88% def-use paths while our proposed approach provides 100% fault detection rate as well as def-use path coverage.

8.3 Validation of Our Proposed Approach

This section is divided into two sub-sections. The first subsection explains the experimental settings of obtained results while discussion of the obtained results is described in the second subsection.

8.3.1 Experimental Settings

The state machine models of 9 different software systems and corresponding generated XML, are given in Appendix A & B, that are used to perform the cross validation of our approach. XML of each software system is given as input one by one to analyze the data flow within states by a number of ants in a number of generations.

The experimental steps described in section 8.2.1 are performed to carry out the validation. The generated optimal paths based on coverage criteria are analyzed with existing technique. The total number of input models, states, and all def-use pairs within each model is shown in table 8.8.

Model No.	Input Models	DU-Pairs	All DU-Paths
1	ECS	13	43
2	TS	9	16
3	ATM	4	6
4	SES	4	5
5	CC	4	4
6	HMS	6	6
7	LMS	5	5
8	POS	6	13
9	DM	5	10

Table 8.8 Number of Input Models and DU-Pairs

8.3.2 Results and Discussion of Validation of Proposed Approach

Table 8.4 describe the limitations of existing techniques and compared it with proposed technique. There are few model based techniques that perform data flow analysis but don't provide complete coverage of faults. From the literature, categorize of faults are identified and existing techniques are evaluated against them. One of the code based technique provide complete du-path and fault coverage. But none of the model based technique provides 100% fault wise coverage. A model based technique doesn't provide du-path and fault coverage due to presence of infeasible paths. One of the technique identified all du-paths within an input model but don't generate test cases to cover them and also mutation testing is not performed. Our technique identified all categorizes of fault within input model and provide maximum fault-wise coverage within input model. When considering general coverage of input models, our proposed technique give maximum state coverage as compared to existing techniques.

As we have seen from literature that the model based approaches that use metaheuristic techniques only generate test cases focusing on transition and state coverage. None of these model-based approaches use metaheuristic approach for data flow testing purpose.

8.4 Comparison

Currently, all the existing approaches and tools are capable of analyzing and performing data flow testing of state based systems based on data flow analysis rules. Our proposed approach also carry out so. However, the major difference lies in the way flow of data within state is analyzed and the coverage criterion is fulfilled with minimum number of test cases. Therefore, instead of focusing on the generated optimal solution, we have based our comparison on the all overall process of analysis of data. This section is dedicated to present a description of our comparison.

8.4.1 Data Flow Testing Approaches

The parameters of comparison and their details are given below.

Comparison with Model-Based Data flow testing Approaches

- *All DU-Paths Coverage*

  Existing techniques don't provide complete coverage of all definition-paths, due to undetection of du-pairs within infeasible paths. One of the techniques analyzes data flow and completely identifies definition-use paths but don't generate test cases to cover them. Keeping in view, our approach provides complete coverage.
- *Automated Test Case Generation*

  Existing techniques generate many redundant test cases to fulfill the coverage criteria. One of the techniques doesn't create test cases to provide coverage while other one considers existing test cases. However, our approach generates minimal number of test cases in fulfilling the coverage criteria.
- *Mutation Testing*

  Existing techniques are not effective in detecting data flow faults. However in our approach, we seeded different data flow faults and approach is effective in detecting faults that are within states.

Comparison with Model-Based Data flow testing Approaches using Metaheuristic Techniques

Most of the techniques performs data flow testing are code based using Genetic algorithm and PSO but all transitions and all-uses coverage criteria was used . One of the techniques produced better results if UML diagram don't contain final states using all transition coverage results large test number of cases for covering all the transitions.

Comparison with Model-Based Data flow testing Approaches using Swarm Intelligence

All the existing approaches using Ant Colony Algorithm only generate test sequences without consideration of data flow analysis. Approaches developed a prototype to tool for the generation of test cases from UML state diagram focusing on all states coverage. None of the approach performs data-flow testing using Ant Colony Algorithm. However our approach, along with automated generation of test cases performs data-flow testing providing full coverage of All DU-Paths. But our approach focuses on ALL du-paths coverage a criterion which is the strong criteria to fulfill the criteria with optimal number of test cases.

8.5 Assessment

This section is devoted to specifically present the potential benefits and limitations of our proposed approach.

8.5.1 Benefits

Our proposed approach offers many benefits over the existing state based data flow testing approaches. These benefits are given below.

Automatic Data Flow Testing Analysis: Our approach performs automated analysis of data flow within states.

Feasible State Connection Information: One of the existing approaches provides information about feasible connection between states but don't create graph to select paths from it. While our approach also provide feasible connection of states and create a graph from them to traverse them by ants.

Selection of Optimal paths: Our approach generates optimum paths to provide the better optimal solution.

Fulfilling Coverage Criteria with Minimum No. of Test Sequences: Our approach provides the maximum all definition-use paths coverage with optimum test cases that the other existing approaches done it with redundant and larger test cases to fulfill the criteria.

Providing Maximum State Coverage with Minimum No. of Paths: All states coverage is mostly used by state based approaches and generate large test data to cover all the states but our approach also provides maximum states coverage with optimum number of paths.

Automatic Detection of Data Flow Errors: Our approach is too much effective in detecting faults that are seeded result in complete detection of these data flow faults.

8.5.2 Limitations

Considerations of all Du-paths within States: This approach performs the data flow analysis within states, analysis of du-paths in transition are not performed.

Categorization of Uses: We have focuses only on uses of variable but don't categorize them as c-uses and p-uses.

Different Generations Results: As this approach relies on the heuristic technique, therefore in multiple generation of the same input model have different optimal solutions. The generation of best optimal solution is not assured in every generation.

Quality of Optimal Solution: As this approach use probabilistic technique to traverse and finds the path from graph. Therefore the coverage of the all def-use paths depends on the technique.

Chapter 9

CONCLUSION & FUTURE WORK

9.1 Introduction

This chapter is dedicated to present the important findings from this dissertation. A summary of conclusions is given in section 9.2. Finally section 9.3 wraps up this dissertation by summarizing some future research direction and areas.

9.2 Conclusion

This work can be regard as a contribution to the study of data flow analysis of state based systems. In the last few years, a number of data flows testing of UML state machine approaches and tools have been contributed to this field by the software researchers. Due to these efforts, automatic data flow testing has become quite grown-up.

Most of the existing data flow testing approaches focused on data flow analysis of code and some of the approaches performed data flow testing of models. Model based data flow testing approaches don't provide coverage of all def-use paths and if provide coverage don't focused on generation of test cases. In reality, automated generation of test data providing complete coverage, non redundant test cases, and handling looping problem in state based testing is a complex task and several other limitation restrain the results. This task has also become complicated when using complex state machine models.

A comprehensive survey of the existing literature reveals that currently there are no approaches that perform data flow testing using Ant Colony Optimization algorithm. None of the existing approach offers a complete coverage of all def-use paths. The researches that provide coverage of all def-use paths don't generate test cases. On the other hand, one of them analyzes data flow coverage of existing test suites and relies on user input to indentify infeasible paths in a model. Some of the approaches that focused on both automated data flow analysis and test case generation using metaheuristic approach result in redundant test cases and infeasible paths. Our work starts from these observations to view automated data flow analysis as one to solve with heuristic technique to come with optimize set of data.

In this thesis, we have presented an approach for automated analysis of data flow of UML state machine. XML of state machine models are generated to given as input to produce the required results. After the system is given input, test cases are generated automatically. Heuristic search algorithm, Ant colony algorithm is used for exploring and reducing the search space. The task of ACO is to search for optimal number of test cases fulfilling coverage criteria corresponding to every input source model. Every solution searched by ACO is evaluated against fitness function that we have tailored for state based data flow testing problem. The fitness value produced by the fitness function indicates the appropriateness of optimal solution selected by ACO for the data flow testing of corresponding input model. Data flow coverage solution with best fitness values are selected as the final optimal solution. The optimal solution searched by ACO is then utilized to generate optimized set of test cases from graph of input source model that not only fulfills coverage criteria while reducing the search space. Our aim to provide data flow coverage with minimal number of test cases results in complete detection of data flow faults.

We implemented this approach in a tool known as data flow generator (DFG). This approach is generic that perform data flow testing of UML state machine models. As a proof of concept, we validated our approach by analyzing the data flow of state model of software systems, as state machine represent the dynamic behavior of system. Our experimental results indicate that 100% automated test case generation is performed providing all def-use paths as well as state coverage by this approach. And too effective in detection of data flow errors within states. Prerequisite of this approach is to use XML of state machine models to given an input to tool generated using UML modeling tool Enterprise Architect.

A comprehensive analysis of our experimental results revealed that our fitness function is intelligent enough to look for the optimal paths in a search space. This is rather impossible in existing state bases testing approaches and is effective in detection of data flow faults. Our evaluation shows that our approach is not only effective in automated generation of test cases, rather it is also capable of detection of data flow faults to ensure correct flow of data through the state machine model. And no expertise is required for the application of this approach. This approach makes the data flow analysis process unproblematic by using heuristic technique.

However, this approach also has some limitations. As state machine model becomes complex, automated test case generation may be time consuming. Since, the generation of optimal solution in every generation is not guaranteed due to use of probabilistic technique.

9.3 Future Work

In this section, we present some guidelines and direction for future work that would be rewarding to examine further.

9.3.1 Improve Optimal Set of Test Cases

Although, the use of ACO as a heuristic search technique produces good results, we will use other search based test case generation algorithms to further improve our optimal solution.

9.3.2 Application to Large Scale models

We have validated our approach it to medium and small sized UML state machine models. However in future, we investigate the effectiveness of this approach by applying it to large scale software models.

9.3.3 Reduce the Test Case Generation Time

Automated test cases are generated in an adequate time at the moment, this time will be further reduce to speed up the automated generation of test cases fulfilling certain coverage criteria. This can be done by either improving the efficiency of ACO or improving the stopping criteria of the heuristic search.

9.3.4 Enhance Data Flow Testing Tool

Presently, it is capable of analyzing data flow information within states of UML state based models and automated generation of test cases from these models. This tool should be enhanced to analyze the data flow in transitions.

9.3.5 Applicability of our Approach to other Data Flow Oriented Coverage Criteria's

Now we use the data flow oriented all def-use paths coverage criteria as a stopping criteria. But we should also focus on using other coverage criteria as wells. In all def-use paths coverage, we just indentify the use of variable but don't further identify the c-use and p-uses of variable.

9.3.6 Application to other Models

Currently, this approach has been applied to state machine models of software system. In future, this approach should be validated to analyze flow of data and generate test sequences from other models such as UML Activity Diagrams, UML sequence Diagrams and UML Class Diagrams.

References

1. XUE Xue-dong,"The basic principle and application of Ant colony optimization algorithm", IEEE Transactions on Software Engineering,2010
2. Kamran Ghani,"Searching for Test Data",The University of York Department of Computer Science, 2009
3. Sanjay Singla,"An Automatic Test Data Generation for Data Flow Coverage Using Soft Computing Approach", International Journal of Research and Reviews in Computer Science (IJRRCS), 2011
4. Mingjie Deng, "Automatic Test Data Generation Model by Combining Dataflow Analysis with Genetic Algorithm", IEEE, 2009
5. Offutt A. Jefferson and Pan Jie. "Employing data flow testing on object-oriented classes", IEE Proceedings, 2001, online no. 20010448
6. Andreas S. Andreou, "An automatic software test-data generation scheme based on data flow criteria and genetic algorithms", 2007, DOI 10.1109/CIT.2007.97
7. Alessandra Cavarra, "Inter-agent data flow analysis of Abstract State Machines", Australian Software Engineering Conference, 2009,
8. Jun Hou, " DFTT4CWS: A Testing Tool for Composite Web Services Based on Data-Flow", Sixth Web Information Systems and Applications Conference, 2009
9. Lionel Briand, "Improving the coverage criteria of UML state machines using data flow analysis",2009, Softw. Test. Verif. Reliab.; 20:177–207
10. Tabinda Waheed,"Data Flow Analysis of UML Action Semantics for Executable Models", Springer-Verlag Berlin Heidelberg, 2008
11. Praveen Ranjan Srivastava, "Optimized Test Sequence Generation from Usage Models using Ant Colony Optimization", International journal of software engineering, 2010
12. Bor-Yuan Tsai, "An Automatic Test Case Generator Derived from State-Based Testing", Department of Information Management, Tamsui Oxford University College,2000
13. Harsh Kumar Dubey, "Automated Data Flow Testing", IEEE, 2012, DOI 978-1-4673-0455-9/12
14. Fevzi Belli, "Event-Based Mutation Testing vs. State-Based Mutation Testing - An Experimental Comparison", 35th IEEE Annual Computer Software and Applications Conference, 2011

15. Praveen Ranjan Srivastava, “Structured Testing Using Ant Colony Optimization” , ACM, 2010, 978-1-4503-0408-5/10/12
16. Cheng Li, “Study on Improved Ant Colony Algorithm of Swarm Intelligence Algorithm”, 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE), 2010
17. Sun Chengmin, “The Overview of Feature Selection Algorithms Based Swarm Intelligence and Rough Set”, Seventh International Conference on Computational Intelligence and Security, 2011
18. Chih-Yao Chien, “A New Method for Handling The Travelling Salesman Problem Based on Parallelized Genetic Ant Colony System”, Proceedings of the Eighth International Conference on Machine Learning and Cybernetics, 2009
19. Bharti Suri, “Analyzing Test Case Selection & Prioritization using ACO”, ACM SIGSOFT Software Engineering Notes, November 2011, Volume 36 Number 6
20. Yogesh Singh, “Test Case Prioritization using Ant Colony Optimization”, ACM SIGSOFT Software Engineering Notes, July 2010 ,Volume 35 Number 4
21. XUE Xue-dong, “The Basic Principle and application of Ant Colony Optimization Algorithm”, IEEE, 2010, 978-1-4244-6936-9/10
22. Raluca Lefticaru, “Automatic State-Based Test Generation Using Genetic Algorithms”, IEEE, 2008 0-7695-3078-8/08
23. Chartchai Doungsa-ard, “Test Data Generation from UML State Machine Diagrams using Gas”, IEEE International Conference on Software Engineering Advances(ICSEA 2007), 2007,0-7695-2937-2/07
24. Ilona Bluemke, “Data flow approach to testing Java programs”, IEEE Fourth International Conference on Dependability of Computer Systems, 2009 ,978-0-7695-3674-3/09
25. Praveen Ranjan Srivastava, “Structured Testing Using Ant Colony Optimization”, ACM , 2010 ,978-1-4503-0408-5/10/12,
26. Hyeon-Jeong Kim, “Deriving Data Depencies from/for UML State Machine Diagrams”, IEEE, 2011, 978-0-7695-4453-3
27. Praveen Ranjan Srivastava, “Automated Software Testing using Metaheuristic Technique Based on Ant Colony Optimization”, IEEE, 2010 DOI 101.1109/ISED.2010.52

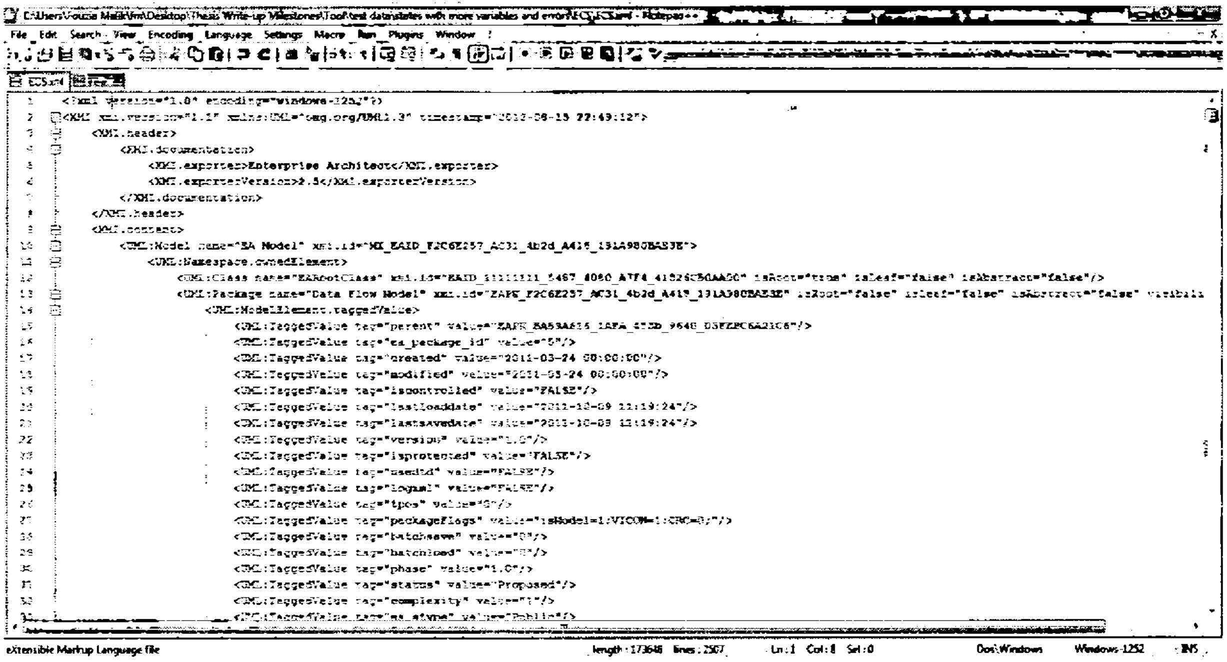
28. AhmedS. Ghiduk,"Using Genetic Algorithms to Aid Test-Data Generation for Data-Flow Coverage",IEEE, 2007
29. Chartchai Doungsa-ard. "Test Data Generation from UML State Machine Diagrams using GA", IEEE International Conference on Software Engineering Advances(ICSEA 2007) , 2007
30. Hua Bai, "A Survey on Application of Swarm Intelligence Computation to Electric Power System",Proceedings of the 6th World Congress on Intelligent Control and Automation, 2006
31. L.C. Briand, "Improving Statechart Testing Criteria Using Data Flow Information", Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05), 2005
32. Moheb R. Girgis, "Automatic Test Data Generation for Data Flow Testing Using a Genetic Algorithm",Journal of Universal Computer Science, 2005
33. Huaizhong LI, "An Ant Colony Optimization Approach to Test Sequence Generation for State based Software Testing ",Proceedings of the Fifth International Conference on Quality Software (QSIC'05), 2005
34. Briand LC, Labiche Y, Wang Y. "Using simulation to empirically investigate test coverage criteria", Proceedings of the IEEE/ACM International Conference on Software Engineering, 2004
35. P. McMinn, "Search-Based Software Test Data Generation: A Survey", Software Testing, Verification and Reliability, 2004
36. G. M. Kapfhammer, "Software testing", The Computer Science Handbook , 2004,Boca Raton,FL: CRC Press
37. M. R. Girgis, "Automatic test data generation for data flow testing using a genetic algorithm", Journal of Universal computer Science, 2005
38. Hyoung Seok Hong,"Data Flow Testing as Model Checking", proceedings of the 25th international conference on Software Engineering (ICSE'03) ,2003
39. Bogdan Korel, "Slicing of State-Based Models", IEEE Proceedings of the International Conference on Software Maintenance (ICSM'03) , 2003
40. "OMG Unified Modeling Language Specification" version 1.3.1, 1st edition 2000

41. Hyoung Seok Hong, “A test sequence selection method for state charts”, Software Testing, Verification and Reliability, Softw. Test. Verif. Reliab. 2000, 10: 203–227
42. Y.G.Kim, “Test cases generation from UML state diagrams”, IEEE Proceeding of Software. 1999, Vol 146. No. 4
43. Sandra Rapps, “Data Flow Analysis Techniques for Test Data Selection”, IEEE, 1982, 0270-5257/82/0000/0272500.75
44. J. R. Horgan, “A Data Flow Coverage Testing Tool for C”, IEEE, 1992 0-8186-2620-8/92
45. Dick Hamlet, “Exploring Dataflow Testing of Arrays”, IEEE, 1993, 0270-5257/93
46. Elaine J. Weyuker, “The Cost of Data Flow Testing: An Empirical Study”,IEEE Transactions on Software Engineering. 1999 ,Vol 16. NO 2
47. Elaine J. Weyuker, “An Empirical Study of the Complexity of Data Flow Testing”, IEEE , 1988,0225-3/88/0000/0188
48. Phyllis ,"An Applicable Family of Data Flow Testing Criteria",IEEE Transactions on Software Engineering, 1998
49. S. Rapps and E. J. Weyuker, "Data flow analysis techniques for test data selection”, Proceedings of the 6th IEEE-CS International Conference on Software Engineering, 1982
50. S. Rapps and E. J. Weyuker, "Selecting software test data using data flow information", IEEE Transactions on Software Engineering, 1985
51. J. Shan, “Research on Formal Description of Data Flow Software Faults”, International Conference on Computer Application and System Modeling (ICCASM 2010), 2010

APPENDIX A

XML Format of All Input Models

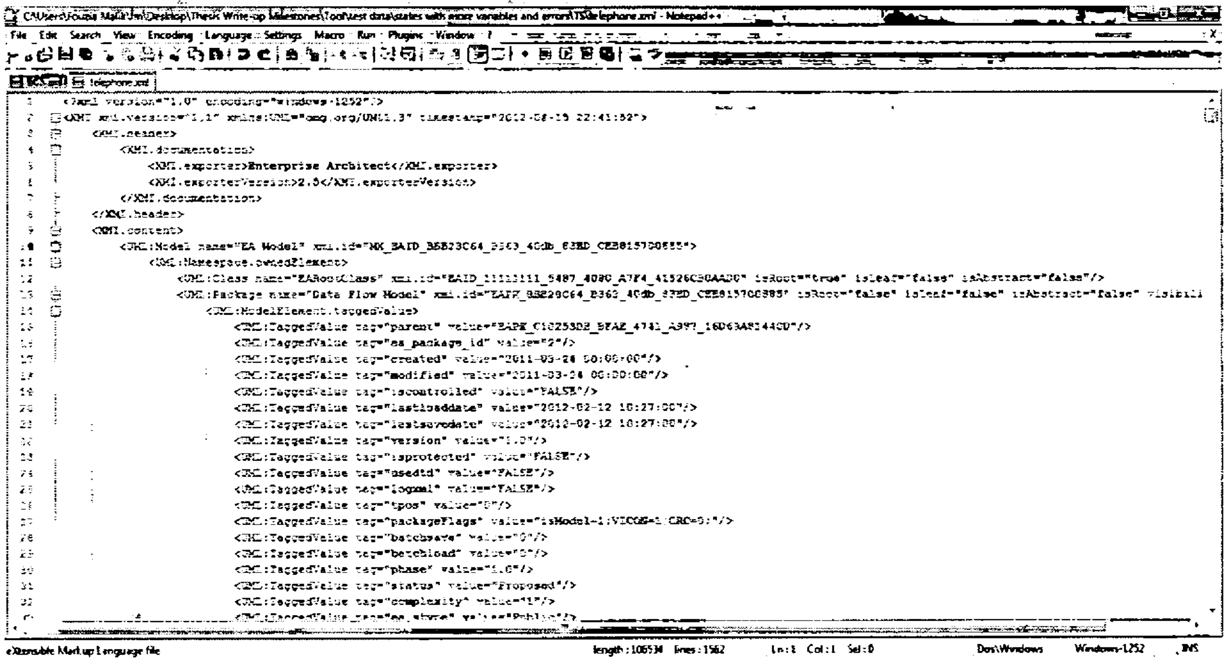
A.1 XML Format of ECS


C:\Users\Fouzia Malik\fm\Desktop\Thesis Write-up Milestones\Tool\test datastates with more variables and error\ECS\ECS.xml - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
ECS.xml

```
<?xml version="1.0" encoding="windows-1252"?>
<XMI xmi.version="1.1" xmlns:UML="omg.org/UML1.3" timestamp="2012-08-15 22:49:12">
    <XMI.header>
        <XMI.documentation>
            <XMI.exporter>Enterprise Architect</XMI.exporter>
            <XMI.exporterVersion>2.5</XMI.exporterVersion>
        </XMI.documentation>
    </XMI.header>
    <XMI.content>
        <UML:Model name="EA Model" xmi.id="MX_EAID_F2C6E257_AC31_4b2d_A415_131A980BAE3E">
            <UML:Namespace.ownedElement>
                <UML:Class name="EARootClass" xmi.id="EAID_11111111_5487_4080_A7F4_41526CB0AA00" isRoot="true" isLeaf="false" isAbstract="false"/>
                <UML:Package name="Data Flow Model" xmi.id="EAPK_F2C6E257_AC31_4b2d_A419_131A980BAE3E" isRoot="false" isLeaf="false" isAbstract="false" visibili
                    <UML:ModelElement.taggedValue>
                        <UML:TaggedValue tag="parent" value="EAPK_BA53A615_1AFA_473b_9648_03FEBC6A21C6"/>
                        <UML:TaggedValue tag="ea_package_id" value="5"/>
                        <UML:TaggedValue tag="created" value="2011-03-24 00:00:00"/>
                        <UML:TaggedValue tag="modified" value="2011-03-24 00:00:00"/>
                        <UML:TaggedValue tag="iscontrolled" value="FALSE"/>
                        <UML:TaggedValue tag="lastloaddate" value="2012-10-09 12:19:24"/>
                        <UML:TaggedValue tag="lastsavedate" value="2012-10-09 12:19:24"/>
                        <UML:TaggedValue tag="version" value="1.0"/>
                        <UML:TaggedValue tag="isprotected" value="FALSE"/>
                        <UML:TaggedValue tag="usedtd" value="FALSE"/>
                        <UML:TaggedValue tag="logxml" value="FALSE"/>
                        <UML:TaggedValue tag="tpos" value="0"/>
                        <UML:TaggedValue tag="packageFlags" value="isModel=1;VICON=1;CRC=0;"/>
                        <UML:TaggedValue tag="batchsave" value="0"/>
                        <UML:TaggedValue tag="batchload" value="0"/>
                        <UML:TaggedValue tag="phase" value="1.0"/>
                        <UML:TaggedValue tag="status" value="Proposed"/>
                        <UML:TaggedValue tag="complexity" value="1"/>
                        <UML:TaggedValue tag="ea_stype" value="Public"/>
```

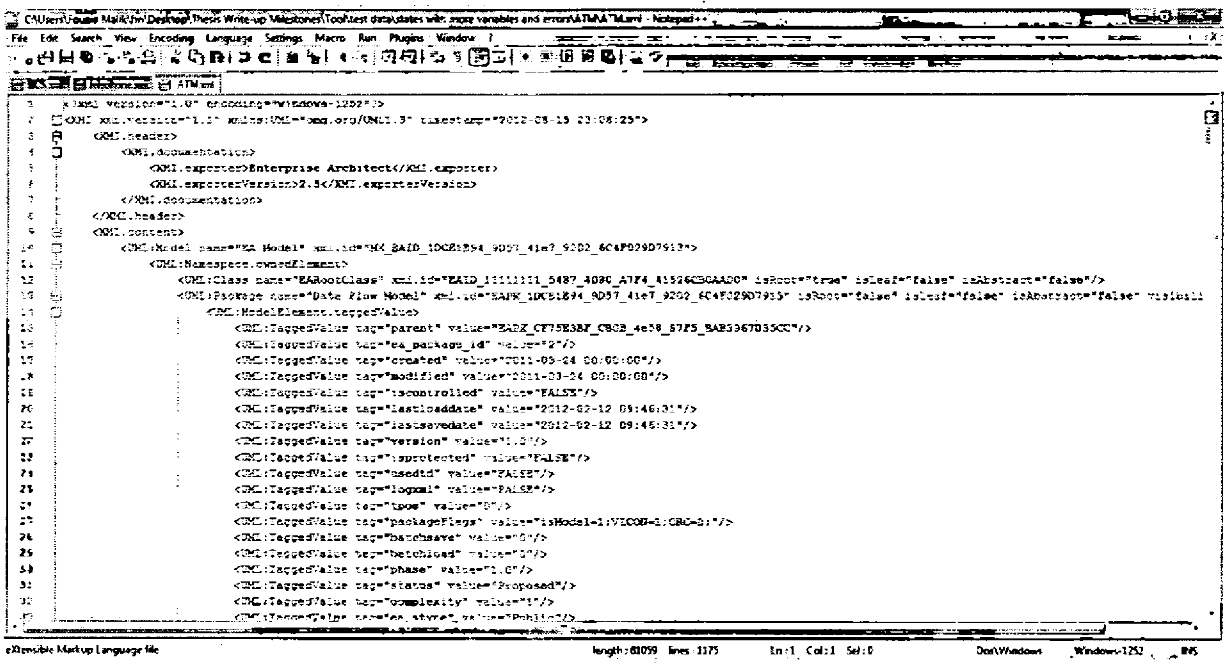
eXtensible Markup Language file   length : 173646   lines : 2507   Ln : 1   Col : 8   Sel : 0   Dos\Windows   Windows-1252   INS

A.2 XML Format of Telephone System


C:\Users\Fouzia Malik\Desktop\Thesis Write-up Milestones\Tool\test data\states with more variables and errors\TS\telephone.xml - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
telephone.xml
eXtensible Markup Language file
length : 106534 lines : 1562
Ln : 1 Col : 1 Sel : 0
Dos\Windows
Windows-1252
INS

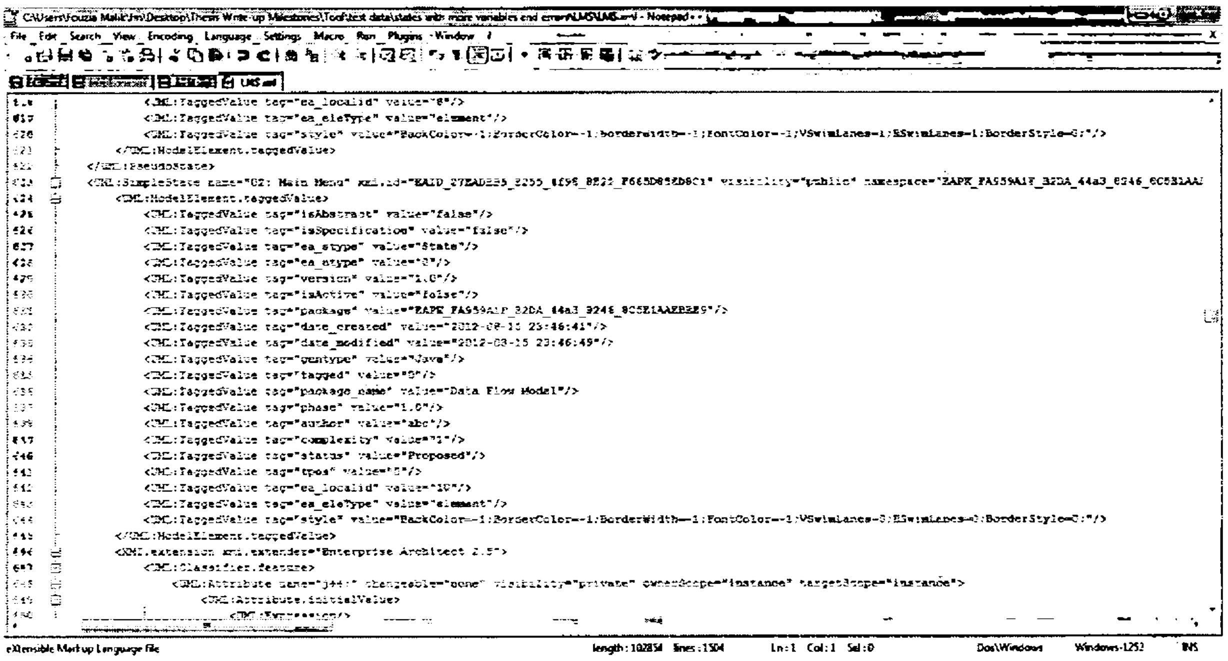
```
<?xml version="1.0" encoding="windows-1252"?>
<XMI xmi.version="1.1" xmlns:UML="omg.org/UML1.3" timestamp="2012-08-19 22:41:52">
    <XMI.header>
        <XMI.documentation>
            <XMI.exporter>Enterprise Architect</XMI.exporter>
            <XMI.exporterVersion>2.5</XMI.exporterVersion>
        </XMI.documentation>
    </XMI.header>
    <XMI.content>
        <UML:Model name="EA Model" xmi.id="MX_EAID_B5B23C64_B363_40db_83BD_CEB0157D0555">
            <UML:Namespace.ownedElement>
                <UML:Class name="EARootClass" xmi.id="EAID_11111111_5487_4080_A7F4_41526CB0AAD0" isRoot="true" isLeaf="false" isAbstract="false"/>
                <UML:Package name="Data Flow Model" xmi.id="EAPK_B5B23C64_B363_40db_83BD_CEB0157D0555" isRoot="false" isLeaf="false" isAbstract="false" visibili
                    <UML:ModelElement.taggedValue>
                        <UML:TaggedValue tag="parent" value="EAPK_C1C253D2_BFAE_4741_A997_16D63A8144CD"/>
                        <UML:TaggedValue tag="ea_package_id" value="2"/>
                        <UML:TaggedValue tag="created" value="2011-03-24 00:00:00"/>
                        <UML:TaggedValue tag="modified" value="2011-03-24 00:00:00"/>
                        <UML:TaggedValue tag="iscontrolled" value="FALSE"/>
                        <UML:TaggedValue tag="lastloaddate" value="2012-02-12 10:27:00"/>
                        <UML:TaggedValue tag="lastsavedate" value="2012-02-12 10:27:00"/>
                        <UML:TaggedValue tag="version" value="1.0"/>
                        <UML:TaggedValue tag="isprotected" value="FALSE"/>
                        <UML:TaggedValue tag="usedtd" value="FALSE"/>
                        <UML:TaggedValue tag="logxml" value="FALSE"/>
                        <UML:TaggedValue tag="tpos" value="0"/>
                        <UML:TaggedValue tag="packageFlags" value="isModel=1;VICON=1;CRC=0;"/>
                        <UML:TaggedValue tag="batchsave" value="0"/>
                        <UML:TaggedValue tag="batchload" value="0"/>
                        <UML:TaggedValue tag="phase" value="1.0"/>
                        <UML:TaggedValue tag="status" value="Proposed"/>
                        <UML:TaggedValue tag="complexity" value="1"/>
                        <UML:TaggedValue tag="ea_stype" value="Public"/>
```

A.3 XML Format of ATM



```xml
<?xml version="1.0" encoding="windows-1252"?>
<XMI xmi.version="1.1" xmlns:UML="omg.org/UML1.3" timestamp="2012-08-15 23:08:25">
    <XMI.header>
        <XMI.documentation>
            <XMI.exporter>Enterprise Architect</XMI.exporter>
            <XMI.exporterVersion>2.5</XMI.exporterVersion>
        </XMI.documentation>
    </XMI.header>
    <XMI.content>
        <UML:Model name="EA Model" xmi.id="MX_EAID_1DCE1894_9D57_41e7_9202_6C4F029D7913">
            <UML:Namespace.ownedElement>
                <UML:Class name="EARootClass" xmi.id="EAID_11111111_5487_4080_A7F4_41526CB0AAD0" isRoot="true" isLeaf="false" isAbstract="false"/>
                <UML:Package name="Data Flow Model" xmi.id="EAPK_1DCE1894_9D57_41e7_9202_6C4F029D7915" isRoot="false" isLeaf="false" isAbstract="false" visibil
                    <UML:ModelElement.taggedValue>
                        <UML:TaggedValue tag="parent" value="EAPK_CF75E3BF_CBCB_4e58_87F5_BAB3967D35CC"/>
                        <UML:TaggedValue tag="ea_package_id" value="2"/>
                        <UML:TaggedValue tag="created" value="2011-03-24 00:00:00"/>
                        <UML:TaggedValue tag="modified" value="2011-03-24 00:00:00"/>
                        <UML:TaggedValue tag="iscontrolled" value="FALSE"/>
                        <UML:TaggedValue tag="lastloaddate" value="2012-02-12 09:46:31"/>
                        <UML:TaggedValue tag="lastsavedate" value="2012-02-12 09:45:31"/>
                        <UML:TaggedValue tag="version" value="1.0"/>
                        <UML:TaggedValue tag="isprotected" value="FALSE"/>
                        <UML:TaggedValue tag="usedtd" value="FALSE"/>
                        <UML:TaggedValue tag="logxml" value="FALSE"/>
                        <UML:TaggedValue tag="tpos" value="0"/>
                        <UML:TaggedValue tag="packageFlags" value="isModel=1;VICON=1;CRC=0;"/>
                        <UML:TaggedValue tag="batchsave" value="0"/>
                        <UML:TaggedValue tag="batchload" value="0"/>
                        <UML:TaggedValue tag="phase" value="1.0"/>
                        <UML:TaggedValue tag="status" value="Proposed"/>
                        <UML:TaggedValue tag="complexity" value="1"/>
                        <UML:TaggedValue tag="ea_stype" value="Public"/>
```

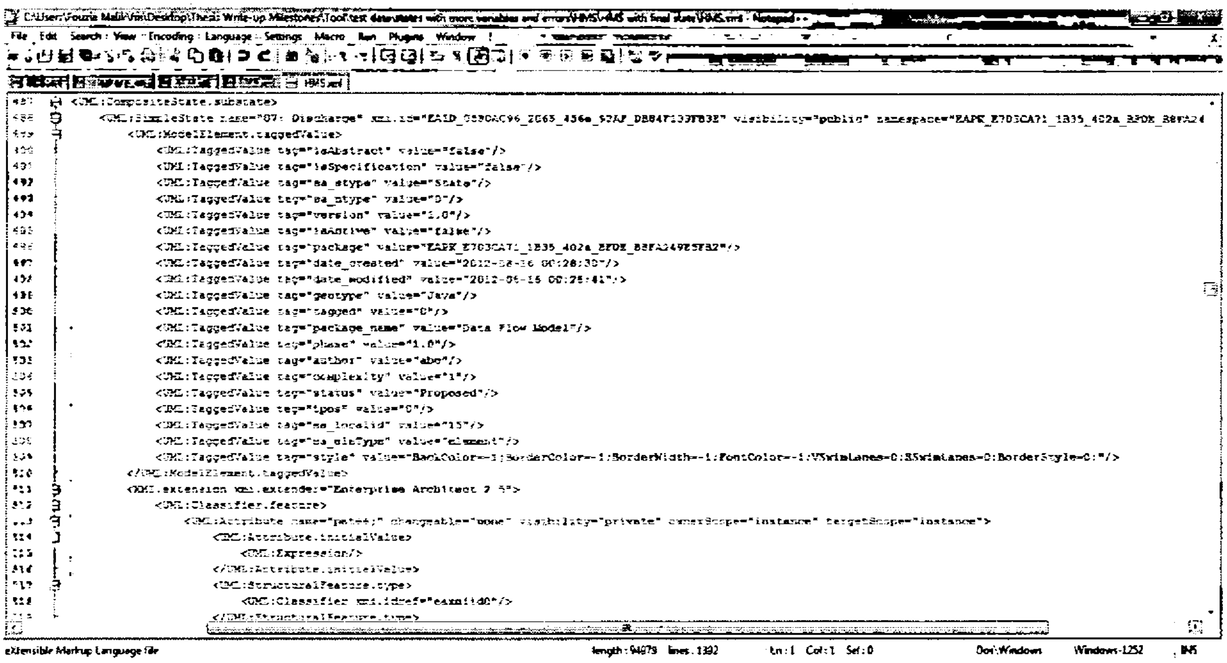
A.4 XML Format of Library Management System


C:\Users\Fouzia Malik\Jm\Desktop\Thesis Write-up Milestones\Tool\test data\states with more variables and error\LMS\LMS.xml - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
LMS.xml

```
                <UML:TaggedValue tag="ea_localid" value="8"/>
                <UML:TaggedValue tag="ea_eleType" value="element"/>
                <UML:TaggedValue tag="style" value="BackColor=-1;BorderColor=-1;BorderWidth=-1;FontColor=-1;VSwimLanes=1;HSwimLanes=1;BorderStyle=0;"/>
            </UML:ModelElement.taggedValue>
        </UML:PseudoState>
        <UML:SimpleState name="02: Main Menu" xmi.id="EAID_27EAD2E5_2255_4f95_8E22_F665D85ED8C1" visibility="public" namespace="EAPK_FA559A1F_32DA_44a3_8246_8C5B1AA
            <UML:ModelElement.taggedValue>
                <UML:TaggedValue tag="isAbstract" value="false"/>
                <UML:TaggedValue tag="isSpecification" value="false"/>
                <UML:TaggedValue tag="ea_stype" value="State"/>
                <UML:TaggedValue tag="ea_ntype" value="0"/>
                <UML:TaggedValue tag="version" value="1.0"/>
                <UML:TaggedValue tag="isActive" value="false"/>
                <UML:TaggedValue tag="package" value="EAPK_FA559A1F_32DA_44a3_8246_8C5B1AAEBEE9"/>
                <UML:TaggedValue tag="date_created" value="2012-08-15 23:46:41"/>
                <UML:TaggedValue tag="date_modified" value="2012-08-15 23:46:49"/>
                <UML:TaggedValue tag="gentype" value="Java"/>
                <UML:TaggedValue tag="tagged" value="0"/>
                <UML:TaggedValue tag="package_name" value="Data Flow Model"/>
                <UML:TaggedValue tag="phase" value="1.0"/>
                <UML:TaggedValue tag="author" value="abc"/>
                <UML:TaggedValue tag="complexity" value="1"/>
                <UML:TaggedValue tag="status" value="Proposed"/>
                <UML:TaggedValue tag="tpos" value="0"/>
                <UML:TaggedValue tag="ea_localid" value="10"/>
                <UML:TaggedValue tag="ea_eleType" value="element"/>
                <UML:TaggedValue tag="style" value="BackColor=-1;BorderColor=-1;BorderWidth=-1;FontColor=-1;VSwimLanes=0;HSwimLanes=0;BorderStyle=0;"/>
            </UML:ModelElement.taggedValue>
            <XMI.extension xmi.extender="Enterprise Architect 2.5">
                <UML:Classifier.feature>
                    <UML:Attribute name="j++;" changeable="none" visibility="private" ownerScope="instance" targetScope="instance">
                        <UML:Attribute.initialValue>
                            <UML:Expression/>
```

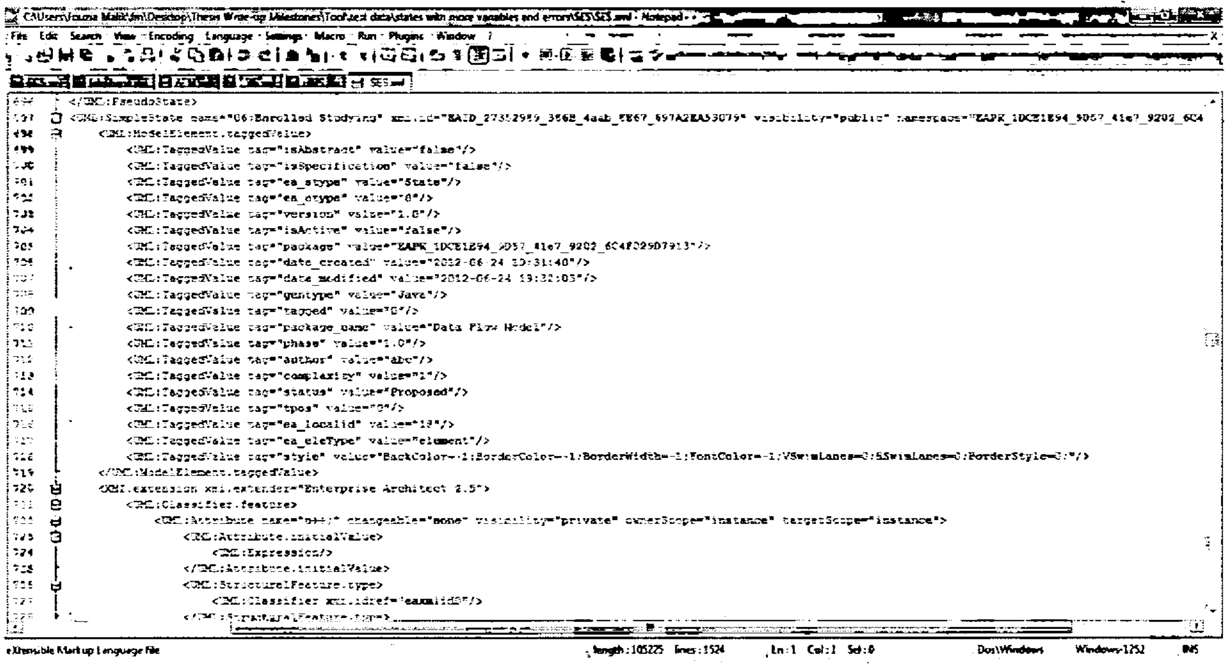
eXtensible Markup Language file   length : 102854   lines : 1504   Ln : 1   Col : 1   Sel : 0   Dos\Windows   Windows-1252   INS

A.5 XML Format of Hotel Management System


C:\Users\Fouzia Malik\fm\Desktop\Thesis Write-up Milestones\Tool\test data\states with more variables and errors\HMS\HMS with final state\HMS.xml - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
HMS.xml
eXtensible Markup Language file
length : 94973 lines : 1392
Ln : 1 Col : 1 Sel : 0
Dos\Windows
Windows-1252
INS

```
<UML:CompositeState.substate>
    <UML:SimpleState name="07: Discharge" xmi.id="EAID_C5BCAC96_2C65_456e_92AF_DB847133FB32" visibility="public" namespace="EAPK_E703CA71_1B35_402a_BFDE_B8FA24
        <UML:ModelElement.taggedValue>
            <UML:TaggedValue tag="isAbstract" value="false"/>
            <UML:TaggedValue tag="isSpecification" value="false"/>
            <UML:TaggedValue tag="ea_stype" value="State"/>
            <UML:TaggedValue tag="ea_ntype" value="0"/>
            <UML:TaggedValue tag="version" value="1.0"/>
            <UML:TaggedValue tag="isActive" value="false"/>
            <UML:TaggedValue tag="package" value="EAPK_E703CA71_1B35_402a_BFDE_B8FA249E5FB2"/>
            <UML:TaggedValue tag="date_created" value="2012-08-16 00:28:30"/>
            <UML:TaggedValue tag="date_modified" value="2012-05-15 00:25:41"/>
            <UML:TaggedValue tag="gentype" value="Java"/>
            <UML:TaggedValue tag="tagged" value="0"/>
            <UML:TaggedValue tag="package_name" value="Data Flow Model"/>
            <UML:TaggedValue tag="phase" value="1.0"/>
            <UML:TaggedValue tag="author" value="abc"/>
            <UML:TaggedValue tag="complexity" value="1"/>
            <UML:TaggedValue tag="status" value="Proposed"/>
            <UML:TaggedValue tag="tpos" value="0"/>
            <UML:TaggedValue tag="ea_localid" value="15"/>
            <UML:TaggedValue tag="ea_eleType" value="element"/>
            <UML:TaggedValue tag="style" value="BackColor=-1;BorderColor=-1;BorderWidth=-1;FontColor=-1;VSwimLanes=0;HSwimLanes=0;BorderStyle=0;"/>
        </UML:ModelElement.taggedValue>
        <XMI.extension xmi.extender="Enterprise Architect 2.5">
            <UML:Classifier.feature>
                <UML:Attribute name="patee;" changeable="none" visibility="private" ownerScope="instance" targetScope="instance">
                    <UML:Attribute.initialValue>
                        <UML:Expression/>
                    </UML:Attribute.initialValue>
                    <UML:StructuralFeature.type>
                        <UML:Classifier xmi.idref="eaxmiid0"/>
                    </UML:StructuralFeature.type>
```

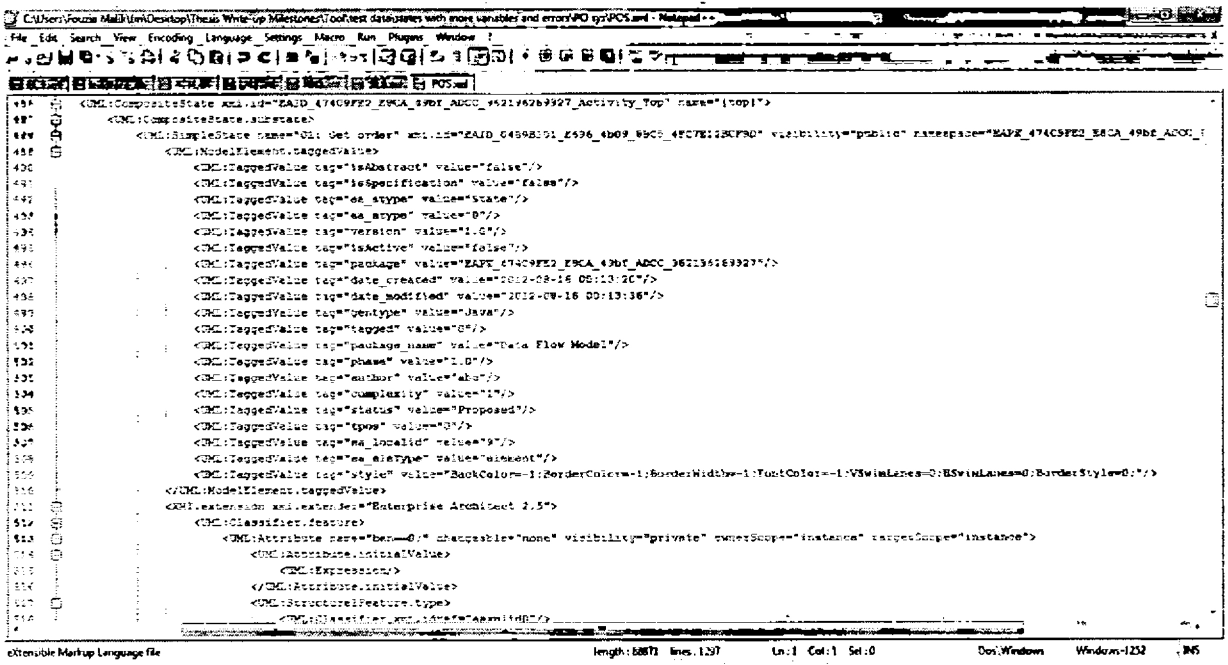
A.6 XML Format of Student Enrollment System


C:\Users\Fouzia Malik\fm\Desktop\Thesis Write-up Milestones\Tool\test data\states with more variables and errors\SES\SES.xml - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
SES.xml

```
</UML:PseudoState>
<UML:SimpleState name="06:Enrolled Studying" xmi.id="EAID_27352989_3862_4aab_8867_697A2EA53079" visibility="public" namespace="EAPK_1DCE1E94_9057_41e7_9202_6C4
    <UML:ModelElement.taggedValue>
        <UML:TaggedValue tag="isAbstract" value="false"/>
        <UML:TaggedValue tag="isSpecification" value="false"/>
        <UML:TaggedValue tag="ea_stype" value="State"/>
        <UML:TaggedValue tag="ea_ntype" value="8"/>
        <UML:TaggedValue tag="version" value="1.0"/>
        <UML:TaggedValue tag="isActive" value="false"/>
        <UML:TaggedValue tag="package" value="EAPK_1DCE1E94_9057_41e7_9202_6C4F2290791 3"/>
        <UML:TaggedValue tag="date_created" value="2012-06-24 13:31:40"/>
        <UML:TaggedValue tag="date_modified" value="2012-06-24 13:32:03"/>
        <UML:TaggedValue tag="gentype" value="Java"/>
        <UML:TaggedValue tag="tagged" value="0"/>
        <UML:TaggedValue tag="package_name" value="Data Flow Model"/>
        <UML:TaggedValue tag="phase" value="1.0"/>
        <UML:TaggedValue tag="author" value="abc"/>
        <UML:TaggedValue tag="complexity" value="1"/>
        <UML:TaggedValue tag="status" value="Proposed"/>
        <UML:TaggedValue tag="tpos" value="0"/>
        <UML:TaggedValue tag="ea_localid" value="19"/>
        <UML:TaggedValue tag="ea_eleType" value="element"/>
        <UML:TaggedValue tag="style" value="BackColor=-1;BorderColor=-1;BorderWidth=-1;FontColor=-1;VSwimLanes=0;HSwimLanes=0;BorderStyle=0;"/>
    </UML:ModelElement.taggedValue>
    <XMI.extension xmi.extender="Enterprise Architect 2.5">
        <UML:Classifier.feature>
            <UML:Attribute name="n++;" changeable="none" visibility="private" ownerScope="instance" targetScope="instance">
                <UML:Attribute.initialValue>
                    <UML:Expression/>
                </UML:Attribute.initialValue>
                <UML:StructuralFeature.type>
                    <UML:Classifier xmi.idref="eaxmiid57"/>
                </UML:StructuralFeature.type>
```

eXtensible Markup Language file   length : 105225   lines : 1524   Ln : 1   Col : 1   Sel : 0   Dos\Windows   Windows-1252   INS

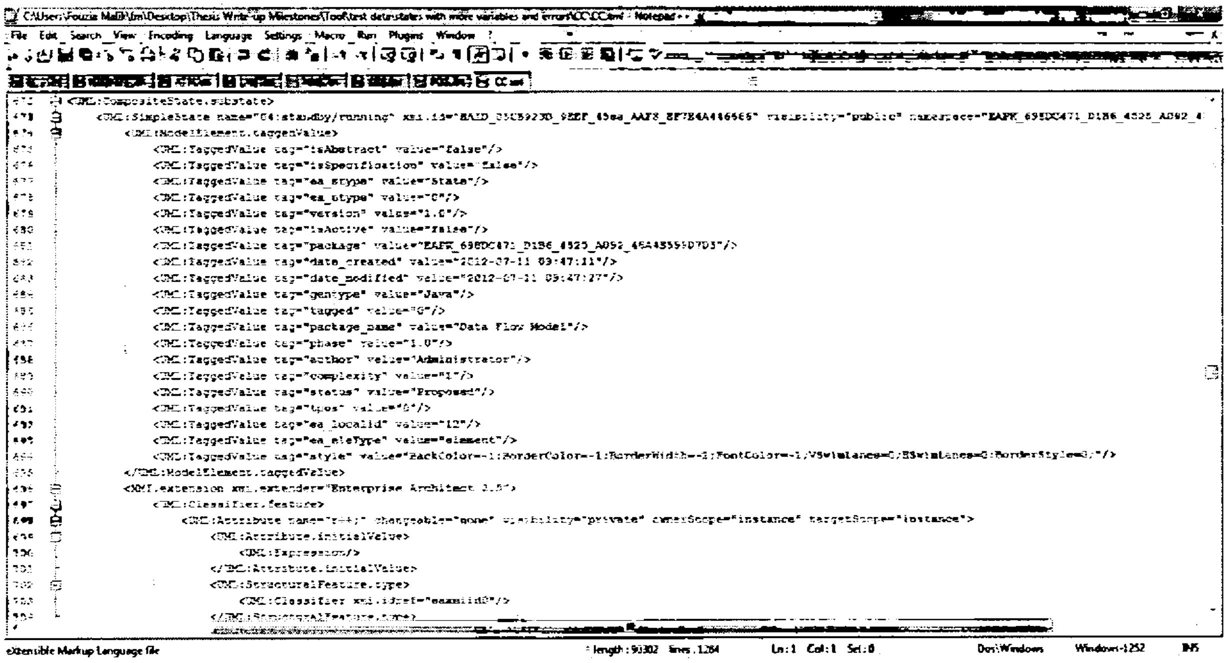
A.7 XML Format of Purchase Order System


C:\Users\Fouzia Malik\fm\Desktop\Thesis Write-up Milestones\Tool\test data\states with more variables and errors\PO sys\POS.xml - Notepad++

```
<UML:CompositeState xmi.id="EAID_474C9FE2_E9CA_49bf_ADCC_9621962B9327_Activity_Top" name="(top)">
    <UML:CompositeState.substate>
        <UML:SimpleState name="01: Get order" xmi.id="EAID_C4598301_E696_4b09_B9C5_4FC7E12BCF9D" visibility="public" namespace="EAPK_474C9FE2_E9CA_49bf_ADCC_
            <UML:ModelElement.taggedValue>
                <UML:TaggedValue tag="isAbstract" value="false"/>
                <UML:TaggedValue tag="isSpecification" value="false"/>
                <UML:TaggedValue tag="ea_stype" value="State"/>
                <UML:TaggedValue tag="ea_ntype" value="0"/>
                <UML:TaggedValue tag="version" value="1.0"/>
                <UML:TaggedValue tag="isActive" value="false"/>
                <UML:TaggedValue tag="package" value="EAPK_474C9FE2_E9CA_49bf_ADCC_9621962B9327"/>
                <UML:TaggedValue tag="date_created" value="2012-08-16 00:13:20"/>
                <UML:TaggedValue tag="date_modified" value="2012-08-16 00:13:36"/>
                <UML:TaggedValue tag="gentype" value="Java"/>
                <UML:TaggedValue tag="tagged" value="0"/>
                <UML:TaggedValue tag="package_name" value="Data Flow Model"/>
                <UML:TaggedValue tag="phase" value="1.0"/>
                <UML:TaggedValue tag="author" value="abc"/>
                <UML:TaggedValue tag="complexity" value="1"/>
                <UML:TaggedValue tag="status" value="Proposed"/>
                <UML:TaggedValue tag="tpos" value="0"/>
                <UML:TaggedValue tag="ea_localid" value="9"/>
                <UML:TaggedValue tag="ea_eleType" value="element"/>
                <UML:TaggedValue tag="style" value="BackColor=-1;BorderColor=-1;BorderWidth=-1;FontColor=-1;VSwimLanes=0;HSwimLanes=0;BorderStyle=0;"/>
            </UML:ModelElement.taggedValue>
            <XMI.extension xmi.extender="Enterprise Architect 2.5">
                <UML:Classifier.feature>
                    <UML:Attribute name="ben=0;" changeable="none" visibility="private" ownerScope="instance" targetScope="instance">
                        <UML:Attribute.initialValue>
                            <UML:Expression/>
                        </UML:Attribute.initialValue>
                        <UML:StructuralFeature.type>
                            <UML:Classifier xmi.idref="eaxmiid0"/>
```

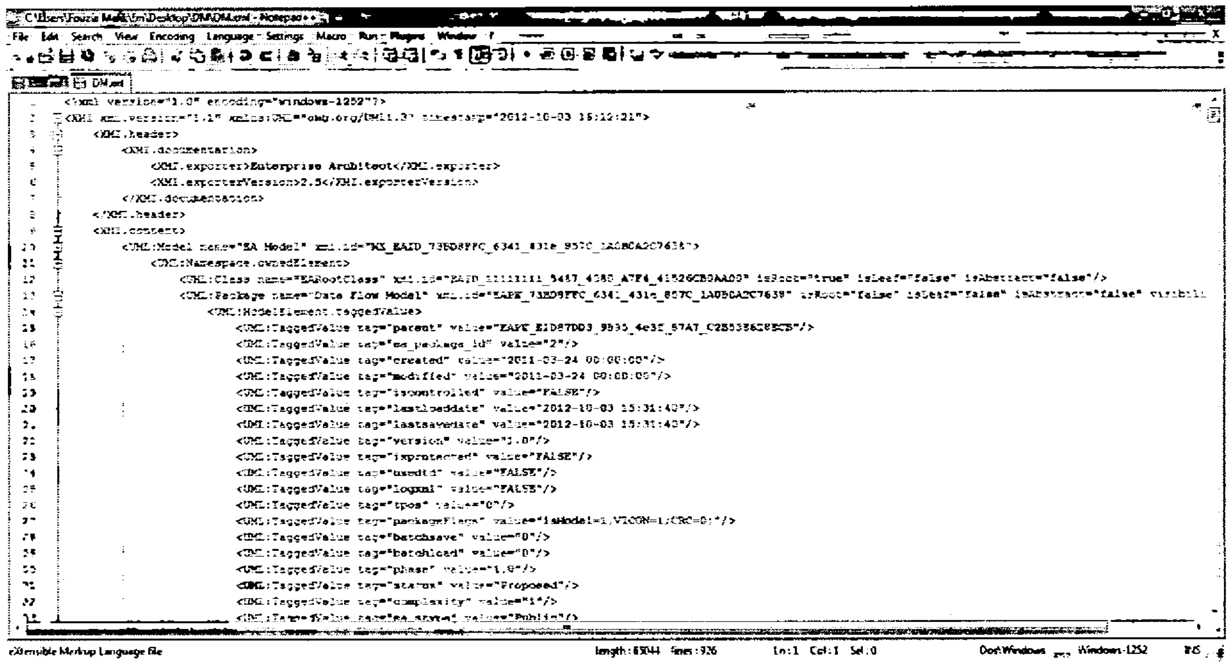
eXtensible Markup Language file   length : 58871   lines : 1297   Ln : 1   Col : 1   Sel : 0   Dos\Windows   Windows-1252   INS

A.8 XML Format of Cruise Control


C:\Users\Fouzia Malik\fm\Desktop\Thesis Write-up Milestones\Tool\test data states with more variables and errors\CC\CC.xml - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
eXtensible Markup Language file
length : 93302 lines : 1264
Ln : 1 Col : 1 Sel : 0
Dos\Windows
Windows-1252
INS

```
<UML:CompositeState.substate>
        <UML:SimpleState name="S4:standby/running" xmi.id="EAID_35C5923D_9EEF_45ea_AAF8_8F7B4A446565" visibility="public" namespace="EAPK_695DC471_D1B6_4525_A092_4
            <UML:ModelElement.taggedValue>
                <UML:TaggedValue tag="isAbstract" value="false"/>
                <UML:TaggedValue tag="isSpecification" value="false"/>
                <UML:TaggedValue tag="ea_stype" value="State"/>
                <UML:TaggedValue tag="ea_ntype" value="0"/>
                <UML:TaggedValue tag="version" value="1.0"/>
                <UML:TaggedValue tag="isActive" value="false"/>
                <UML:TaggedValue tag="package" value="EAPK_695DC471_D1B6_4525_A092_45A435590703"/>
                <UML:TaggedValue tag="date_created" value="2012-07-11 09:47:11"/>
                <UML:TaggedValue tag="date_modified" value="2012-07-11 09:47:27"/>
                <UML:TaggedValue tag="gentype" value="Java"/>
                <UML:TaggedValue tag="tagged" value="0"/>
                <UML:TaggedValue tag="package_name" value="Data Flow Model"/>
                <UML:TaggedValue tag="phase" value="1.0"/>
                <UML:TaggedValue tag="author" value="Administrator"/>
                <UML:TaggedValue tag="complexity" value="1"/>
                <UML:TaggedValue tag="status" value="Proposed"/>
                <UML:TaggedValue tag="tpos" value="0"/>
                <UML:TaggedValue tag="ea_localid" value="12"/>
                <UML:TaggedValue tag="ea_eleType" value="element"/>
                <UML:TaggedValue tag="style" value="BackColor=-1;BorderColor=-1;BorderWidth=-1;FontColor=-1;VSwimLanes=0;HSwimLanes=0;BorderStyle=0;"/>
            </UML:ModelElement.taggedValue>
            <XMI.extension xmi.extender="Enterprise Architect 2.5">
                <UML:Classifier.feature>
                    <UML:Attribute name="r++;" changeable="none" visibility="private" ownerScope="instance" targetScope="instance">
                        <UML:Attribute.initialValue>
                            <UML:Expression/>
                        </UML:Attribute.initialValue>
                        <UML:StructuralFeature.type>
                            <UML:Classifier xmi.idref="eaxmiid0"/>
                        </UML:StructuralFeature.type>
```

A.9 XML Format of Display Manager



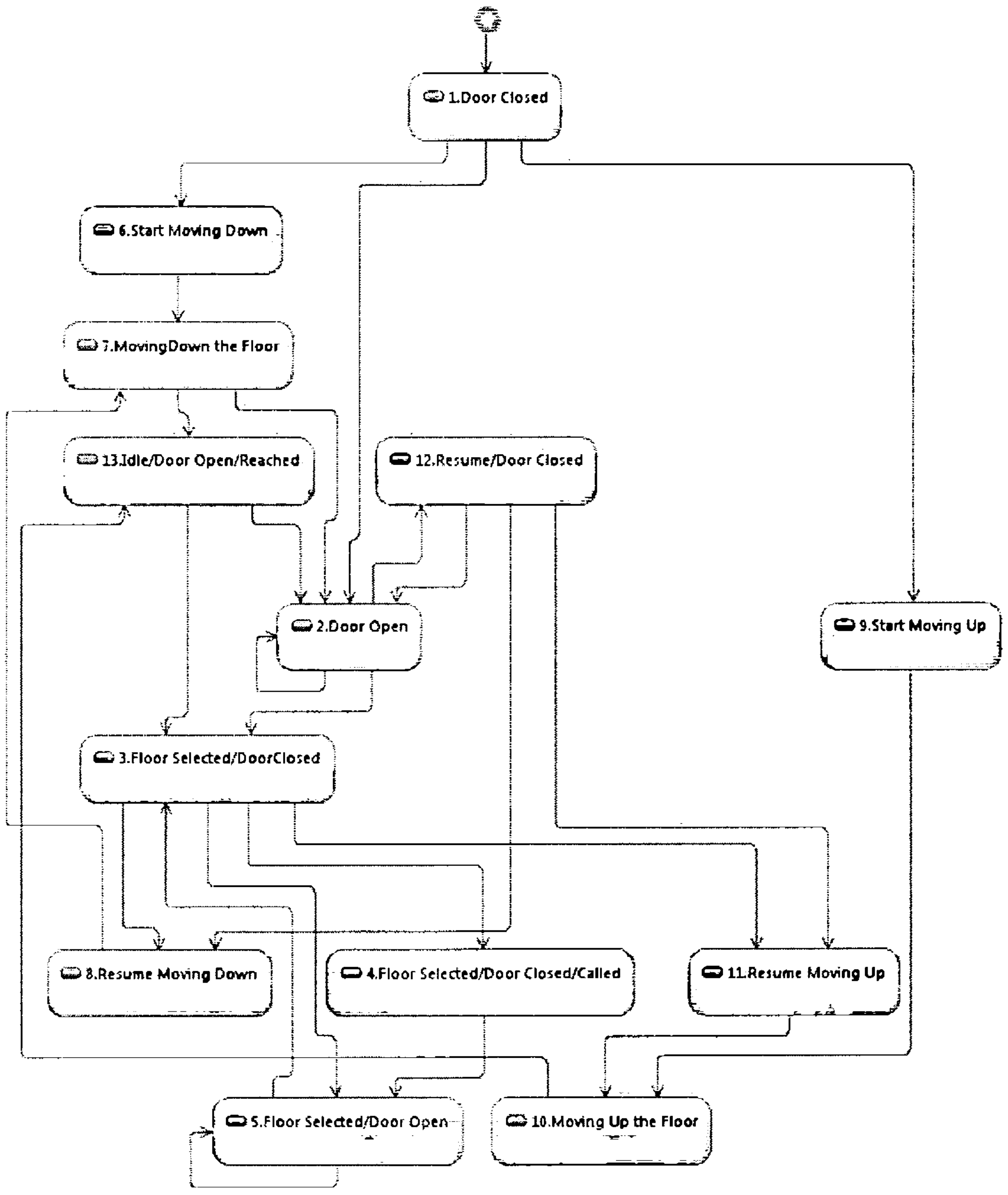
```
<?xml version="1.0" encoding="windows-1252"?>
<XMI xmi.version="1.1" xmlns:UML="omg.org/UML1.3" timestamp="2012-10-03 15:12:21">
    <XMI.header>
        <XMI.documentation>
            <XMI.exporter>Enterprise Architect</XMI.exporter>
            <XMI.exporterVersion>2.5</XMI.exporterVersion>
        </XMI.documentation>
    </XMI.header>
    <XMI.content>
        <UML:Model name="EA Model" xmi.id="MX_EAID_73ED8FFC_6341_431e_857C_1A0BCA2C7638">
            <UML:Namespace.ownedElement>
                <UML:Class name="EARootClass" xmi.id="EAID_11111111_5487_4080_A7F4_41526CB0AA00" isRoot="true" isLeaf="false" isAbstract="false"/>
                <UML:Package name="Data Flow Model" xmi.id="EAPK_73ED8FFC_6341_431e_857C_1A0BCA2C7638" isRoot="false" isLeaf="false" isAbstract="false" visibili
                    <UML:ModelElement.taggedValue>
                        <UML:TaggedValue tag="parent" value="EAPK_E1D87DD3_9535_4e3E_57A7_C255356E8ECB"/>
                        <UML:TaggedValue tag="ea_package_id" value="2"/>
                        <UML:TaggedValue tag="created" value="2011-03-24 00:00:00"/>
                        <UML:TaggedValue tag="modified" value="2011-03-24 00:00:00"/>
                        <UML:TaggedValue tag="iscontrolled" value="FALSE"/>
                        <UML:TaggedValue tag="lastloaddate" value="2012-10-03 15:31:40"/>
                        <UML:TaggedValue tag="lastsavedate" value="2012-10-03 15:31:40"/>
                        <UML:TaggedValue tag="version" value="1.0"/>
                        <UML:TaggedValue tag="isprotected" value="FALSE"/>
                        <UML:TaggedValue tag="usedtd" value="FALSE"/>
                        <UML:TaggedValue tag="logxml" value="FALSE"/>
                        <UML:TaggedValue tag="tpos" value="0"/>
                        <UML:TaggedValue tag="packageFlags" value="isModel=1;VICON=1;CRC=0;"/>
                        <UML:TaggedValue tag="batchsave" value="0"/>
                        <UML:TaggedValue tag="batchload" value="0"/>
                        <UML:TaggedValue tag="phase" value="1.0"/>
                        <UML:TaggedValue tag="status" value="Proposed"/>
                        <UML:TaggedValue tag="complexity" value="1"/>
                        <UML:TaggedValue tag="ea_stype" value="Public"/>
```

C:\Users\Fouzia Malik\fm\Desktop\DM\DM.xml - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
DM.xml
eXtensible Markup Language file
length : 65044 lines : 926
Ln : 1 Col : 1 Sel : 0
Dos\Windows
Windows-1252
INS

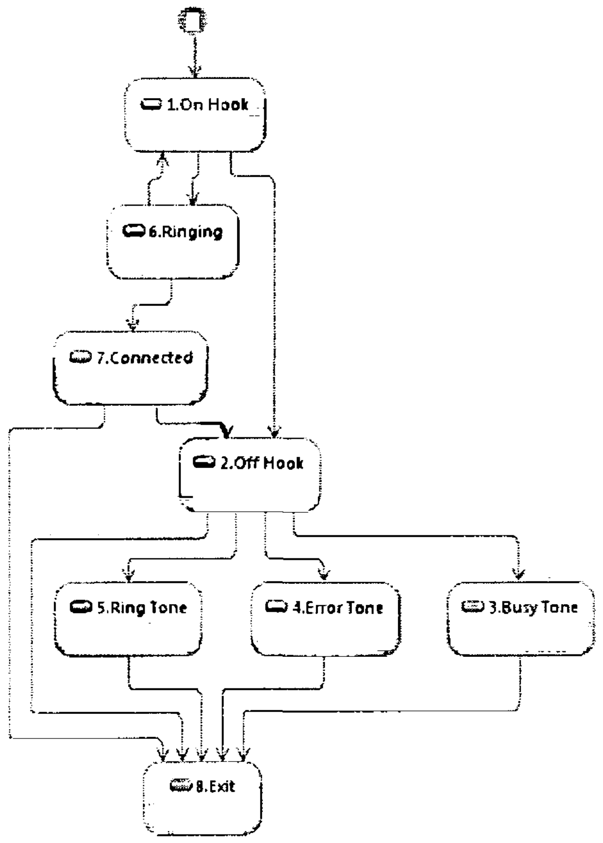
APPENDIX B

Screen Shots of All Input UML State Machine Models

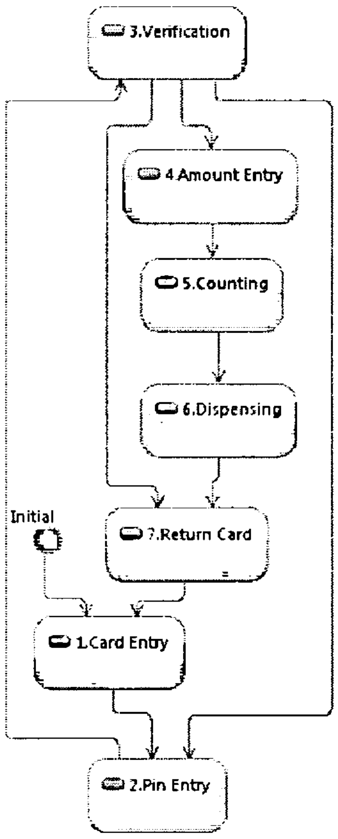
B.1 Screen Shot of ECS



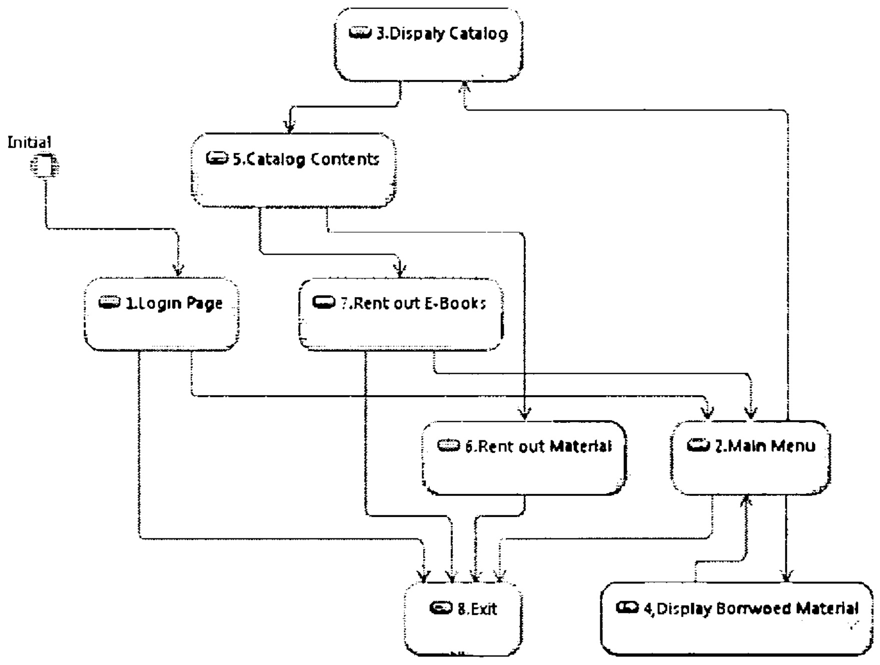
B.2 Screen Shot of Telephone System



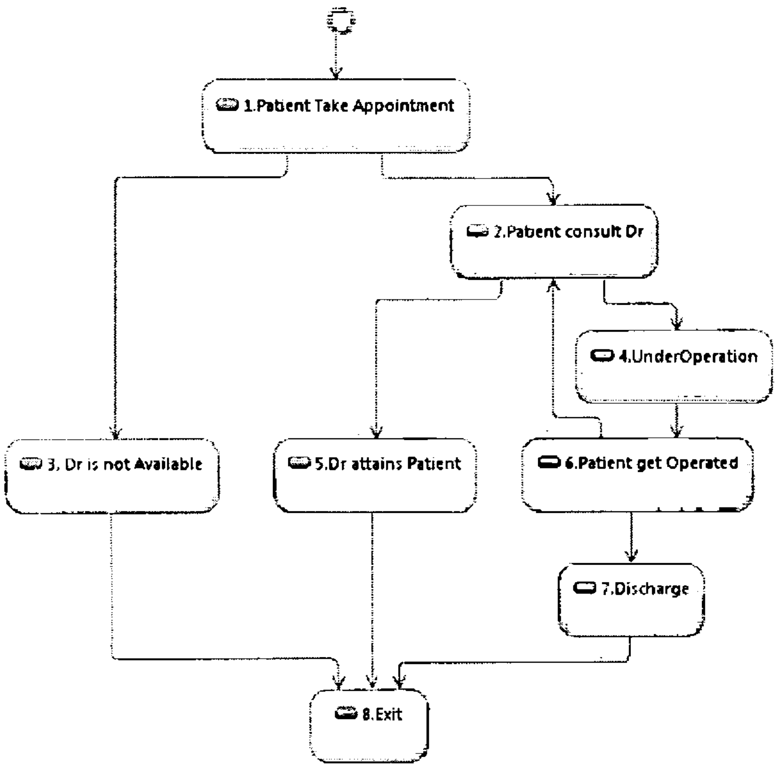
B.3 Screen Shot of ATM



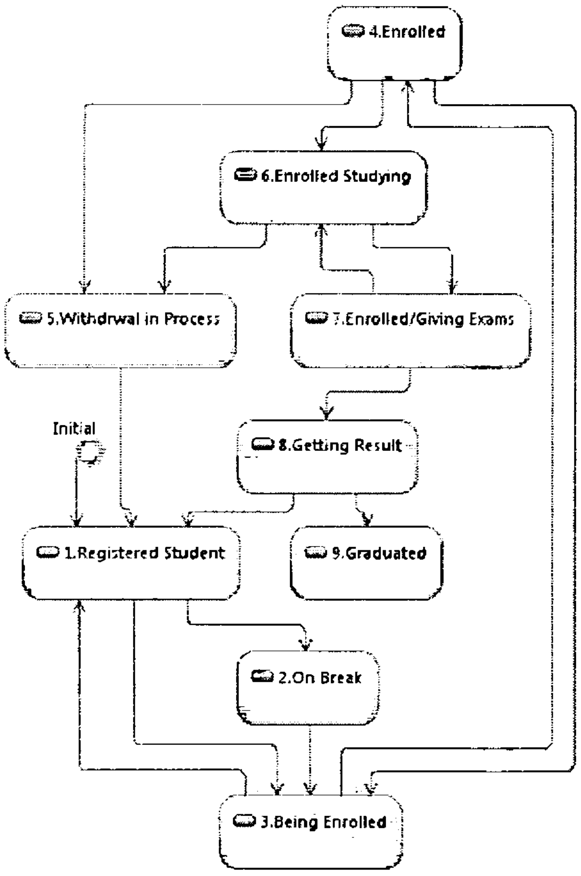
B.4 Screen Shot of Library Management System



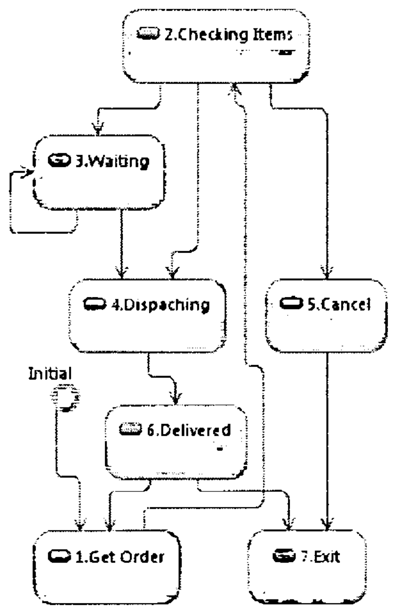
B.5 Screen Shot of Hotel Management System



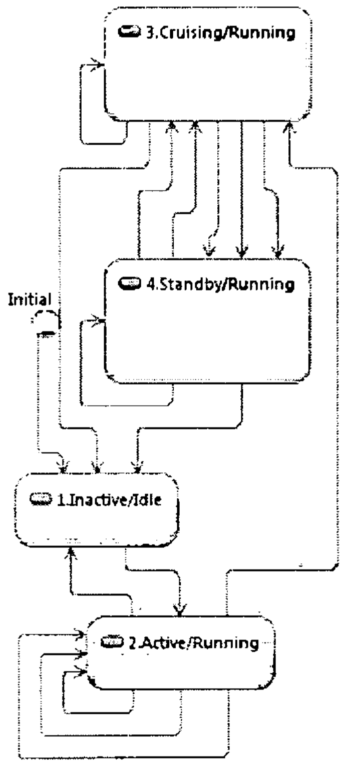
B.6 Screen Shot of Student Enrollment System



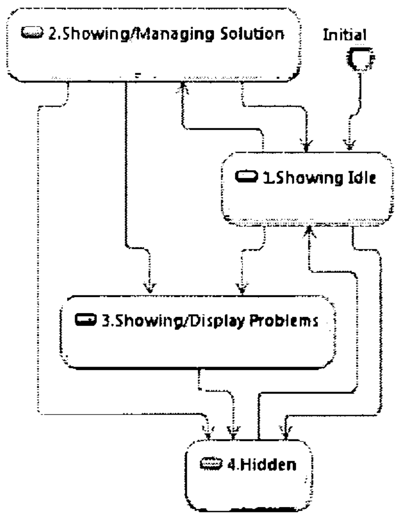
B.7 Screen Shot of Purchase Order System



B.8 Screen Shot of Cruise Control



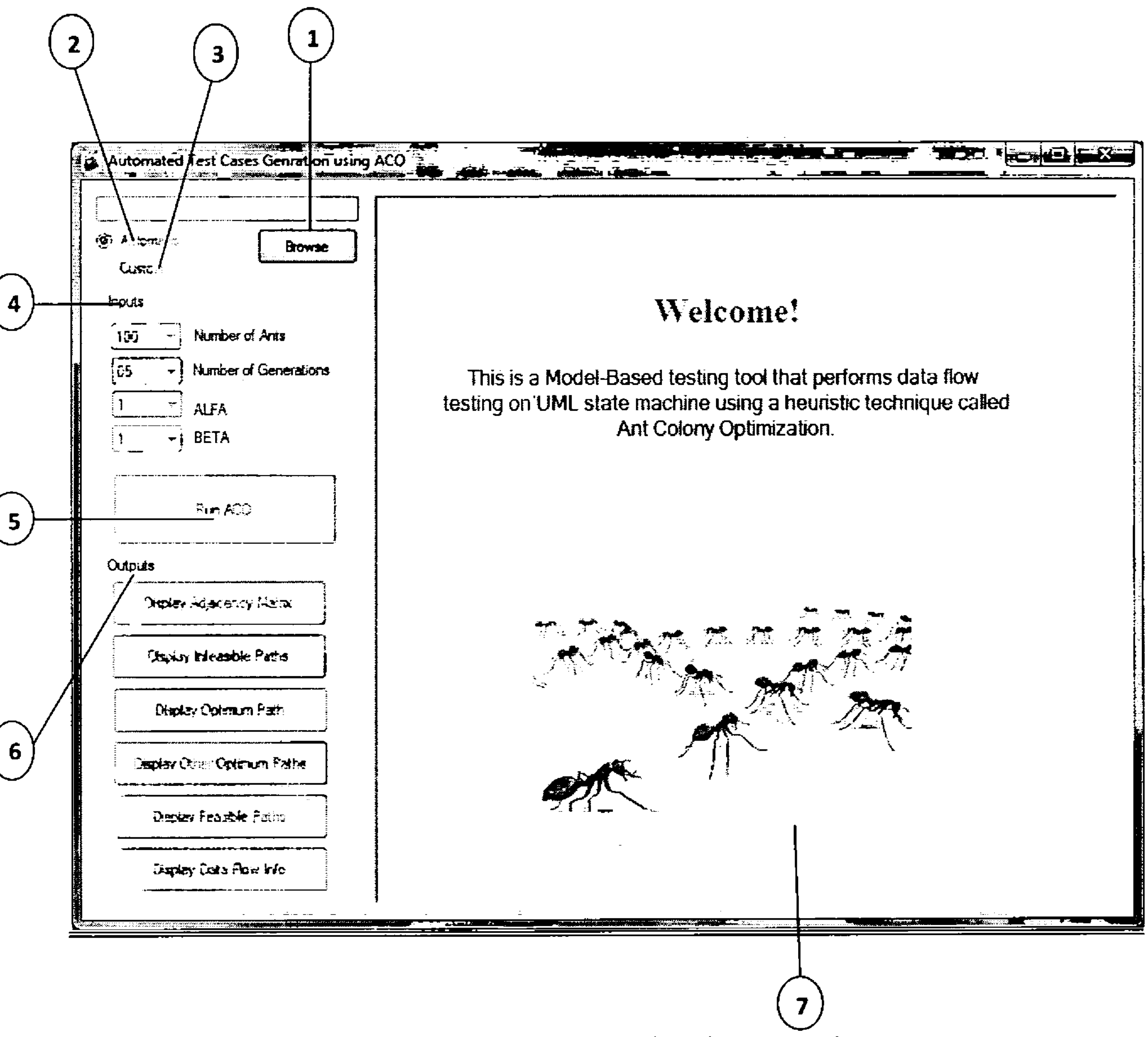
B.9 Screen Shot of Display Manager



APPENDIX C

User Manual

C.1 Main Interface of Data Flow Testing Tool



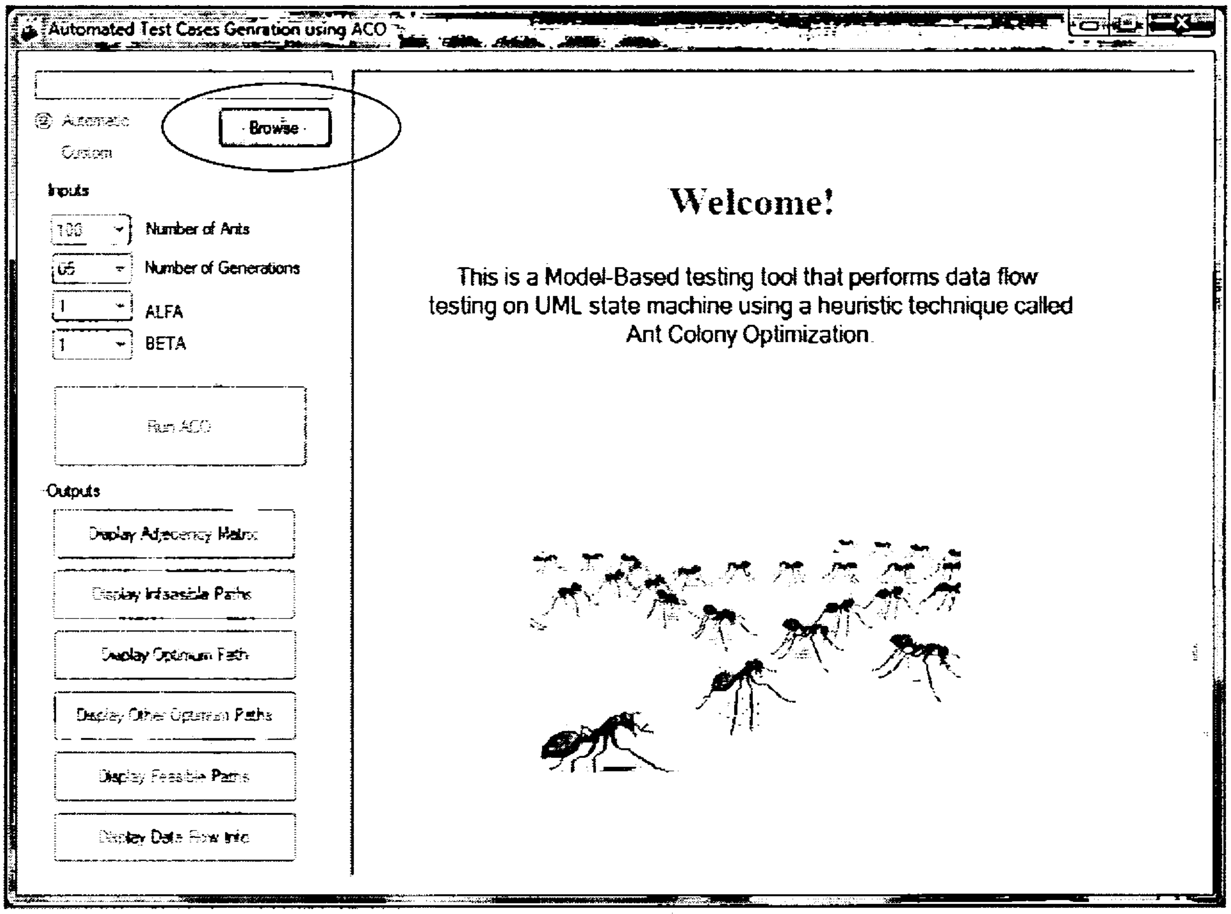
100 Number of Ants
05 Number of Generations
1 ALFA
1 BETA

Welcome!

This is a Model-Based testing tool that performs data flow testing on UML state machine using a heuristic technique called Ant Colony Optimization.

1. **Browse-** Select XML to given input to tool.
2. **Automatic Setting-** Select this option to automatically select parameters values.
3. **Custom Setting-** Select this option to select parameters by users according to his own will.
4. **Inputs Setting-** This option show drop down menu to select input by user when selection custom setting.
5. **RUN ACO** Select this option to apply ACO on given input.
6. **Select Output to Display-** Select different outputs to displays.
7. **Output Window-** This window displays all the Outputs.

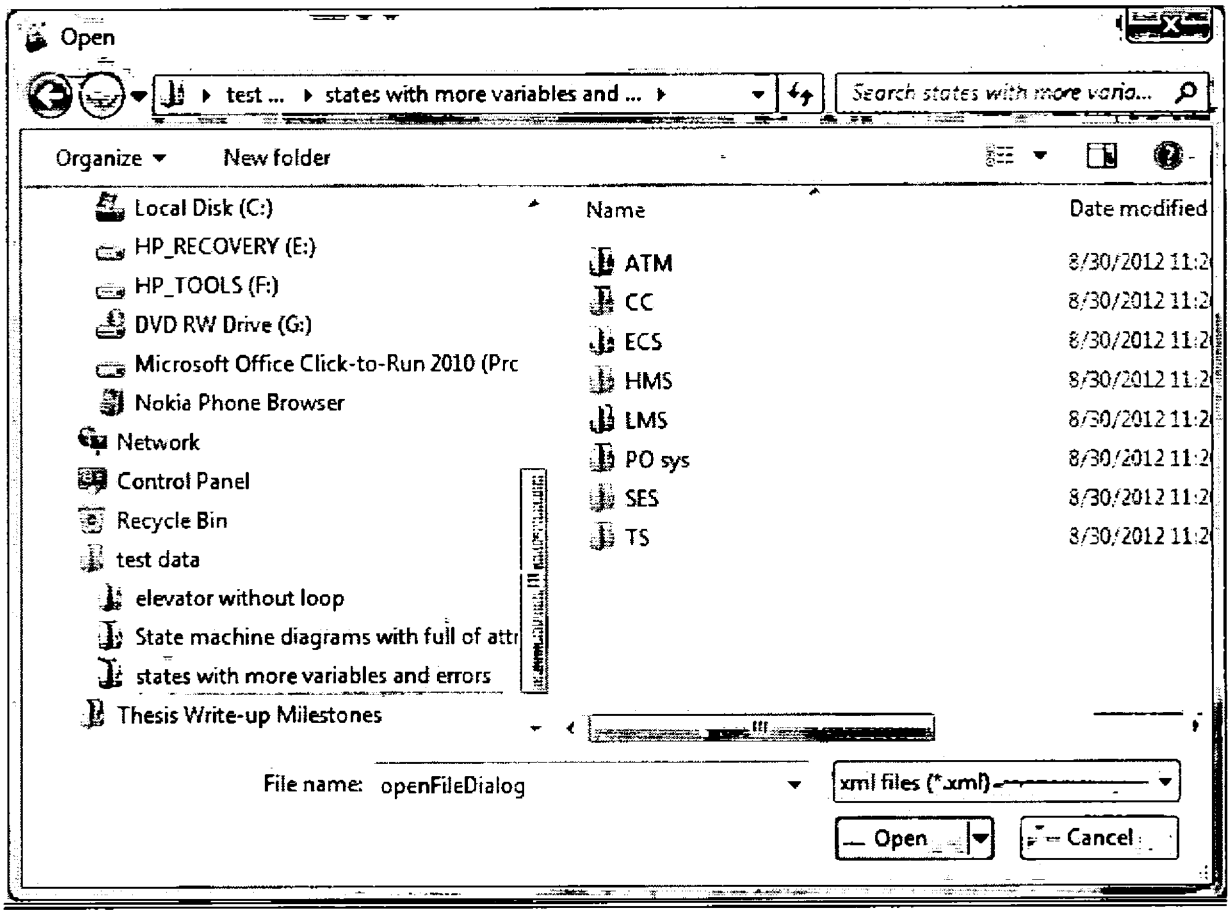
C.2 Browse XML (New XML file)



Welcome!

This is a Model-Based testing tool that performs data flow testing on UML state machine using a heuristic technique called Ant Colony Optimization.

New XML file Window

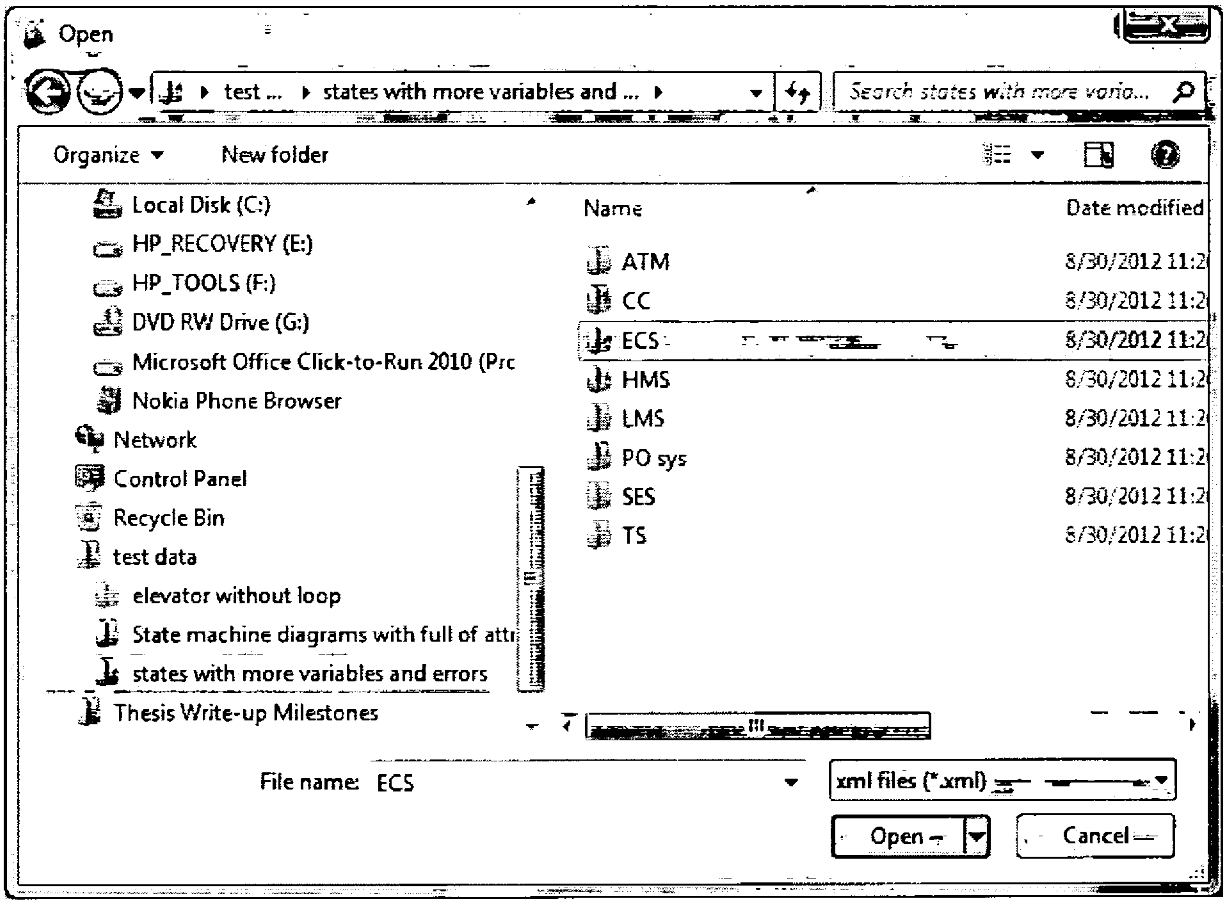

Open
test ... ▸ states with more variables and ... ▸
Search states with more varia...
Organize
New folder
Local Disk (C:)
HP_RECOVERY (E:)
HP_TOOLS (F:)
DVD RW Drive (G:)
Microsoft Office Click-to-Run 2010 (Prc
Nokia Phone Browser
Network
Control Panel
Recycle Bin
test data
elevator without loop
State machine diagrams with full of attr
states with more variables and errors
Thesis Write-up Milestones
Open
Cancel

Name	Date modified
ATM	8/30/2012 11:2
CC	8/30/2012 11:2
ECS	8/30/2012 11:2
HMS	8/30/2012 11:2
LMS	8/30/2012 11:2
PO sys	8/30/2012 11:2
SES	8/30/2012 11:2
TS	8/30/2012 11:2

File name: openFileDialog

xml files (*.xml)

Select Model


Open

test ... ▸ states with more variables and ... ▸

Search states with more varia...

Organize ▾ New folder

Local Disk (C:)
HP_RECOVERY (E:)
HP_TOOLS (F:)
DVD RW Drive (G:)
Microsoft Office Click-to-Run 2010 (Prc
Nokia Phone Browser
Network
Control Panel
Recycle Bin
test data
elevator without loop
State machine diagrams with full of attr
states with more variables and errors
Thesis Write-up Milestones

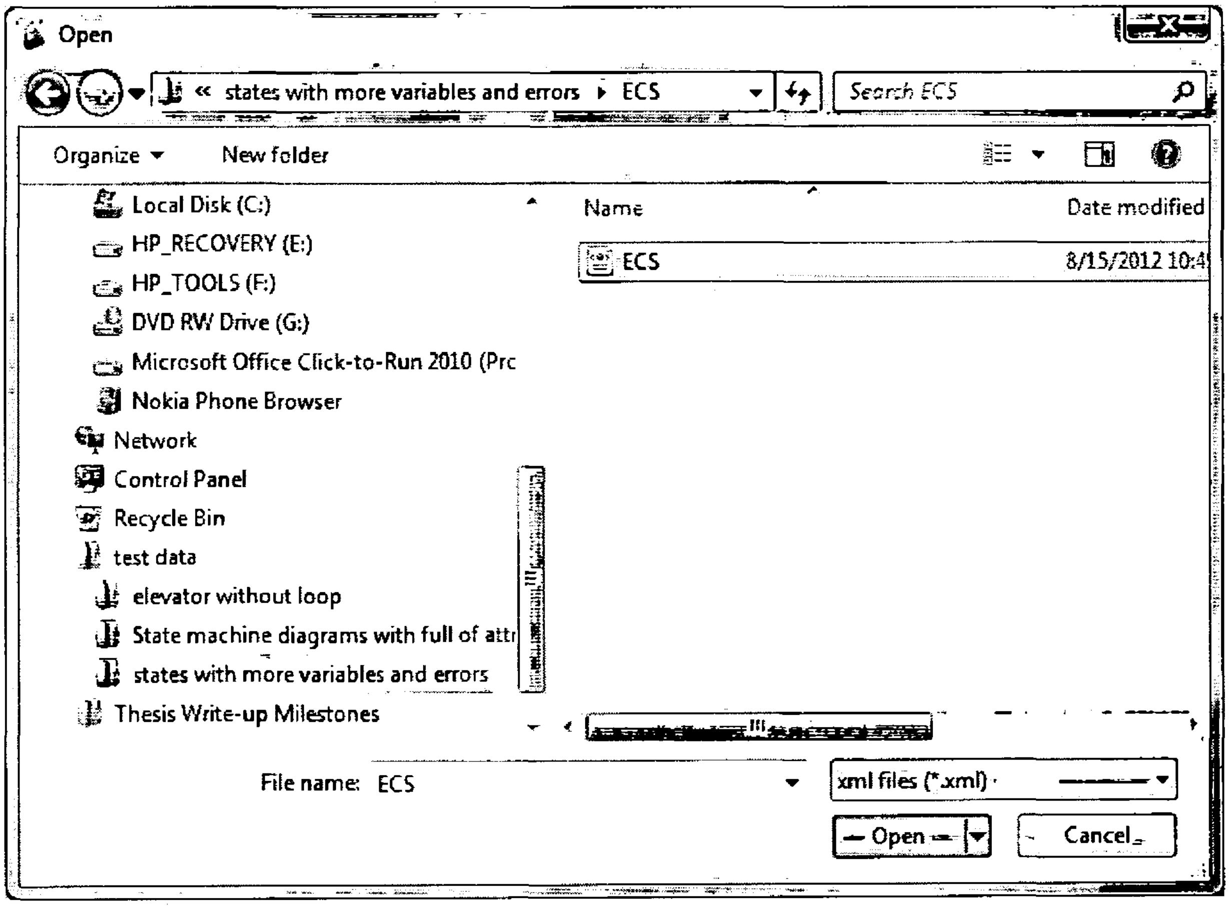
Name	Date modified
ATM	8/30/2012 11:2
CC	8/30/2012 11:2
ECS	8/30/2012 11:2
HMS	8/30/2012 11:2
LMS	8/30/2012 11:2
PO sys	8/30/2012 11:2
SES	8/30/2012 11:2
TS	8/30/2012 11:2

File name: ECS

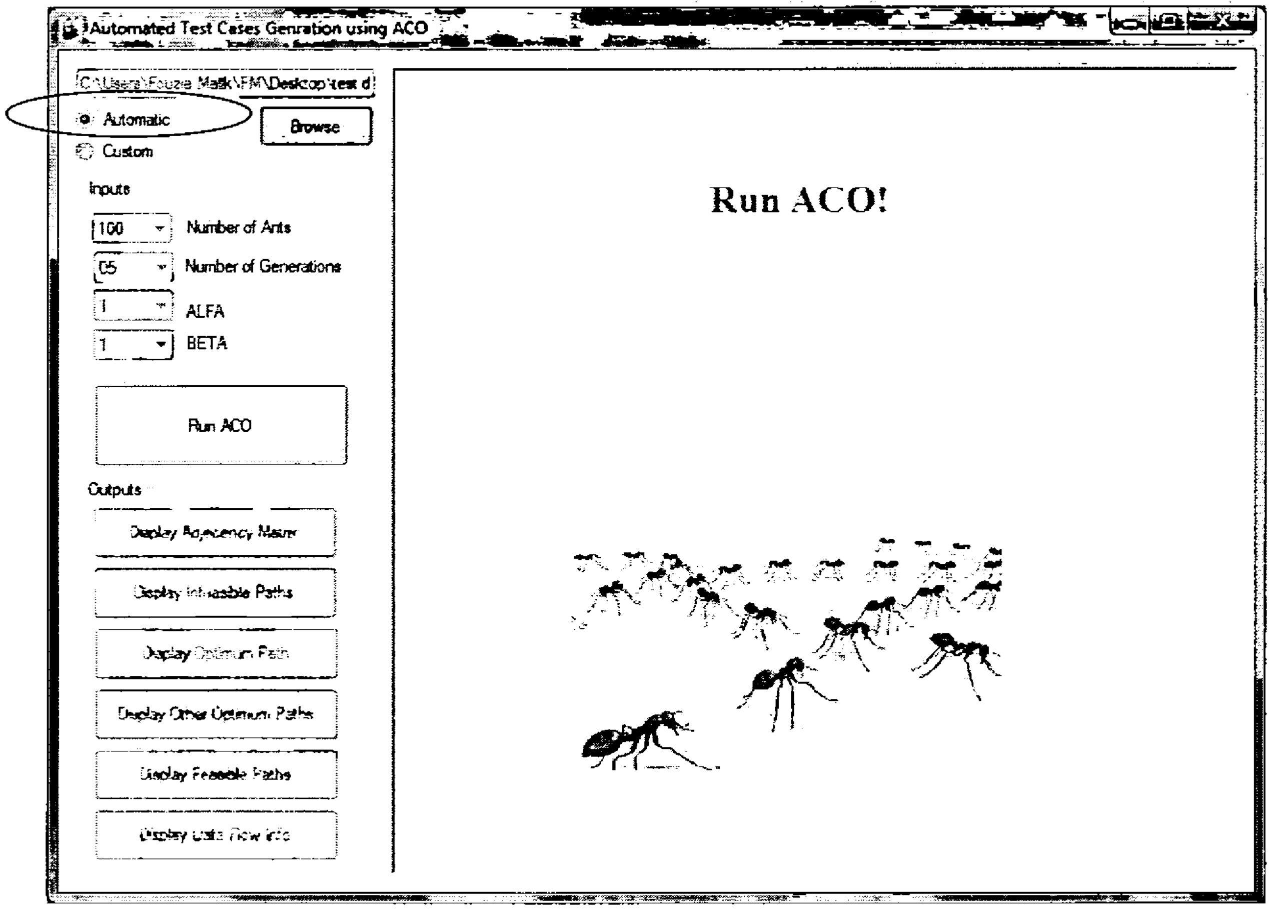
xml files (*.xml)

Open ▾ Cancel

Open XML File of Selected Input Model



C.3 Select Automatic Parameter Setting of ECS Model


Automated Test Cases Genration using ACO

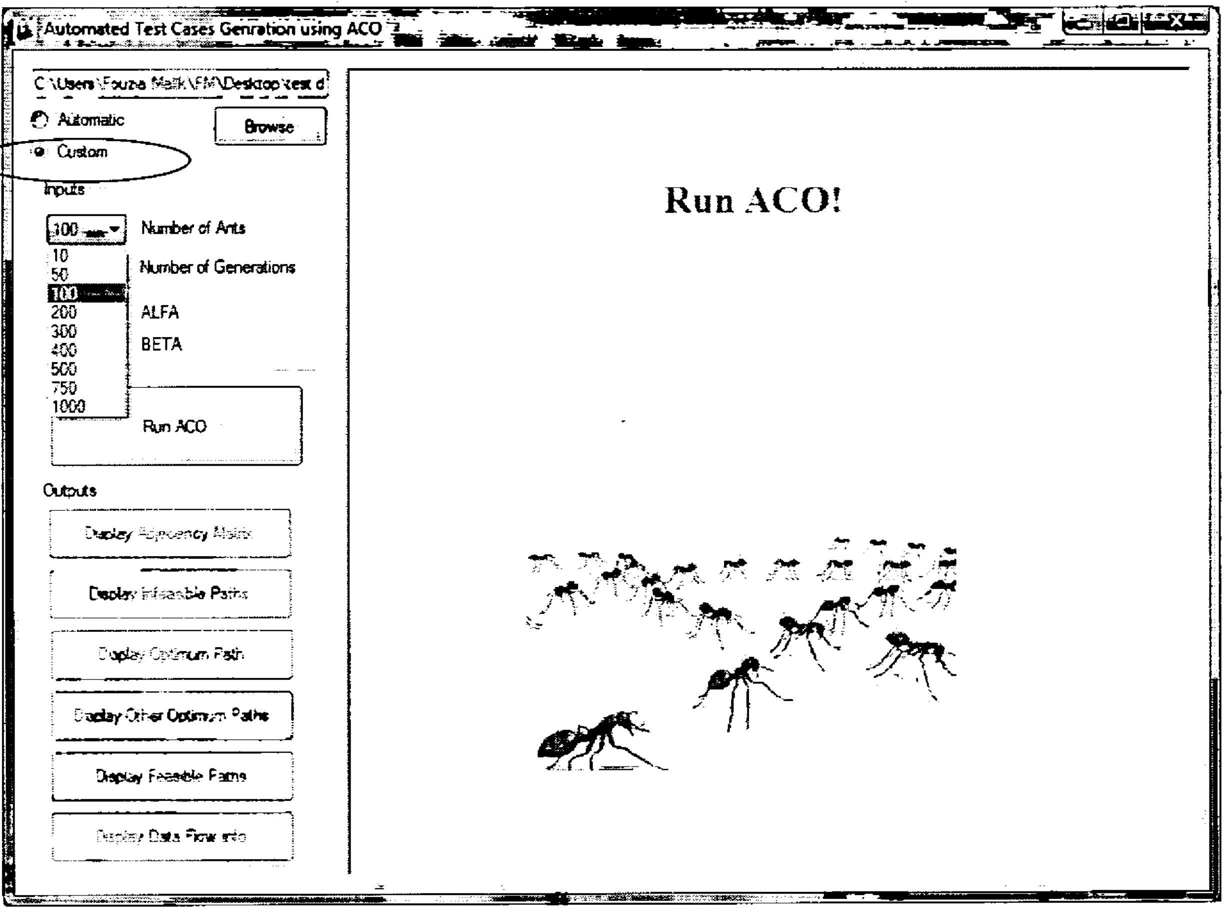
C:\Users\Fouzia Malik\FM\Deskop\test d

[✓ Automatic]
Custom

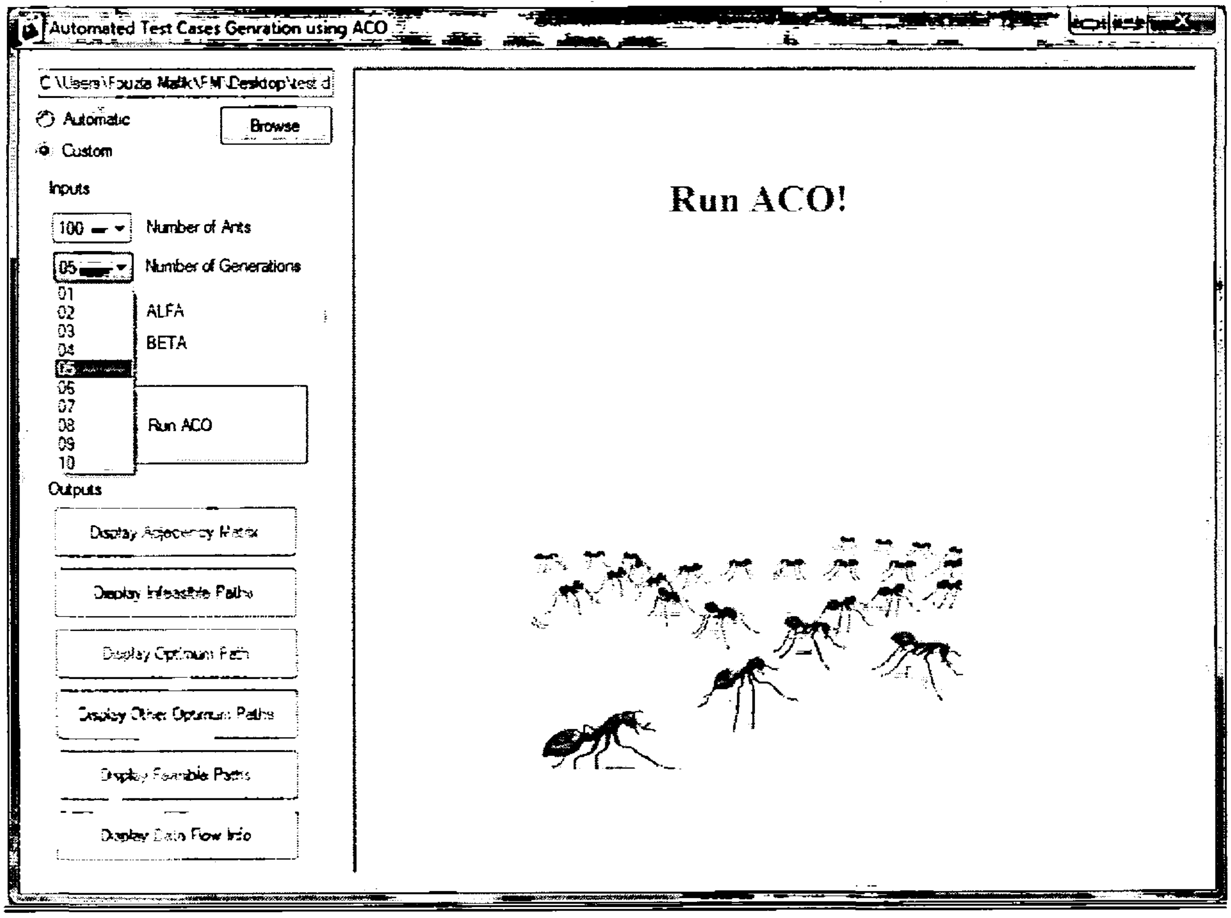
Inputs

Number of Ants: 100
Number of Generations: 05
ALFA: 1
BETA: 1

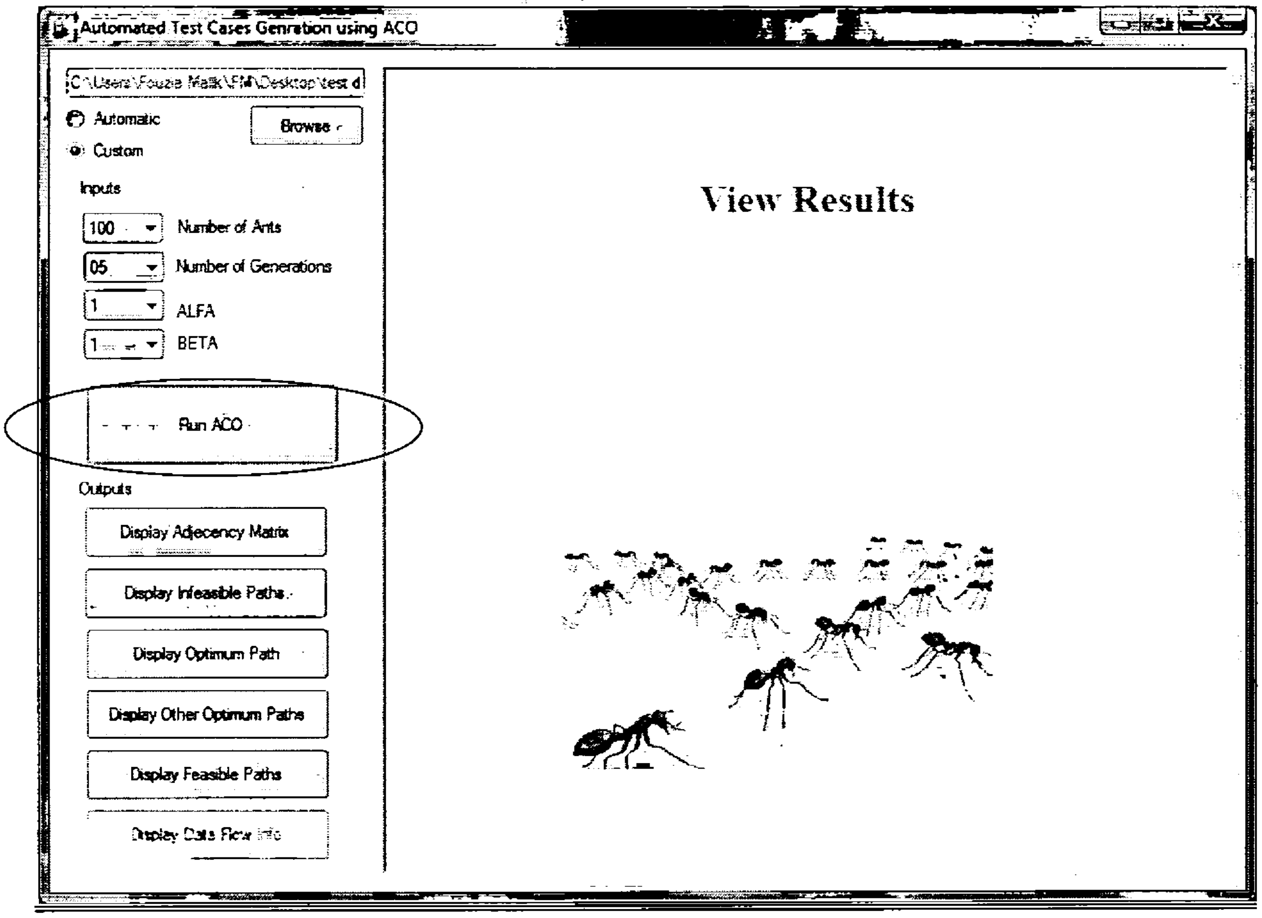
C.4 Select Custom Parameter Setting of ECS Model



Select Number of Generations for ECS Model



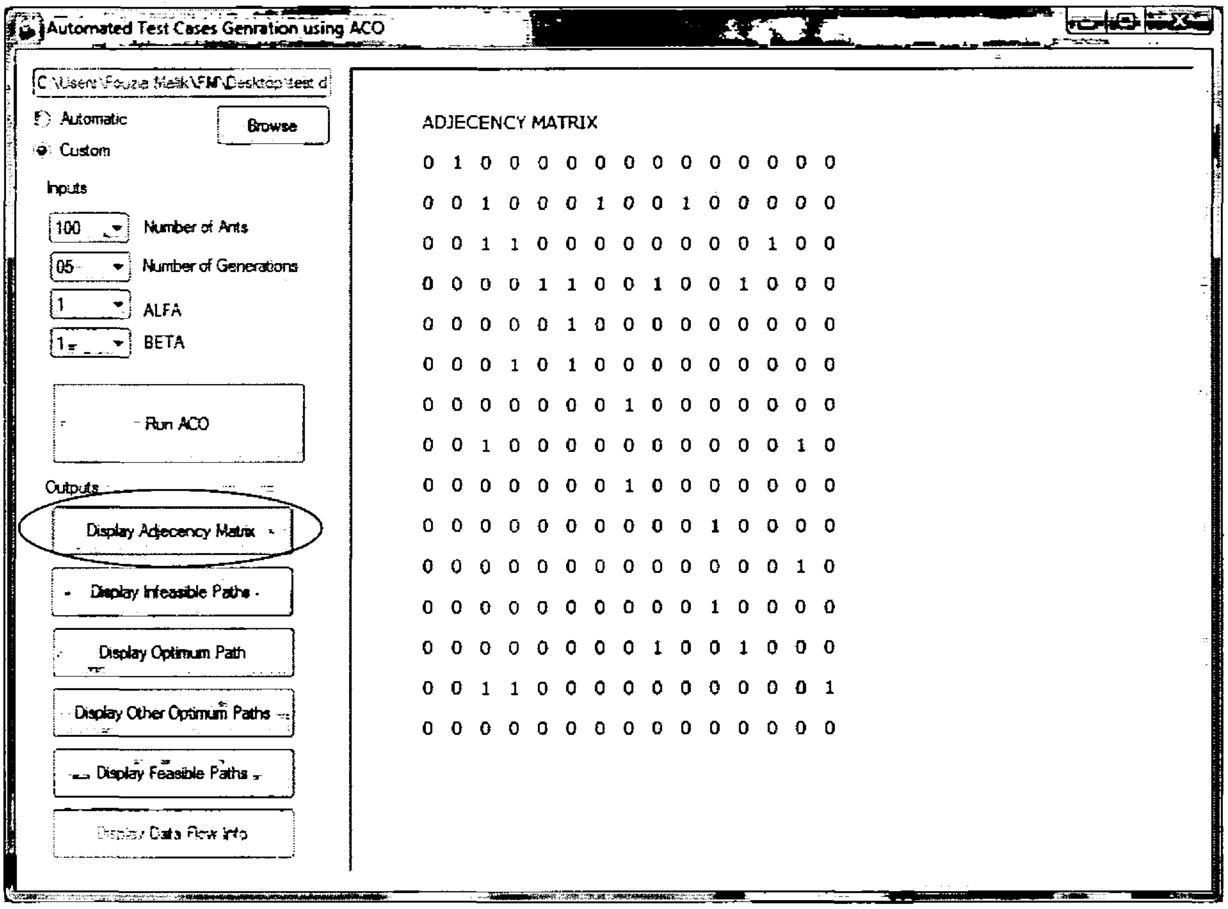
C.4 Run ACO to View the Results



APPENDIX D

Generated Test Cases & Data Flow Information of All Models

D.1 Display Adjacency Matrix of ECS Model



Automated Test Cases Genration using ACO

C:\Users\Fouzia Malik\FM\Desktop\test d

Automatic

[✓ Custom]

Inputs

100 Number of Ants

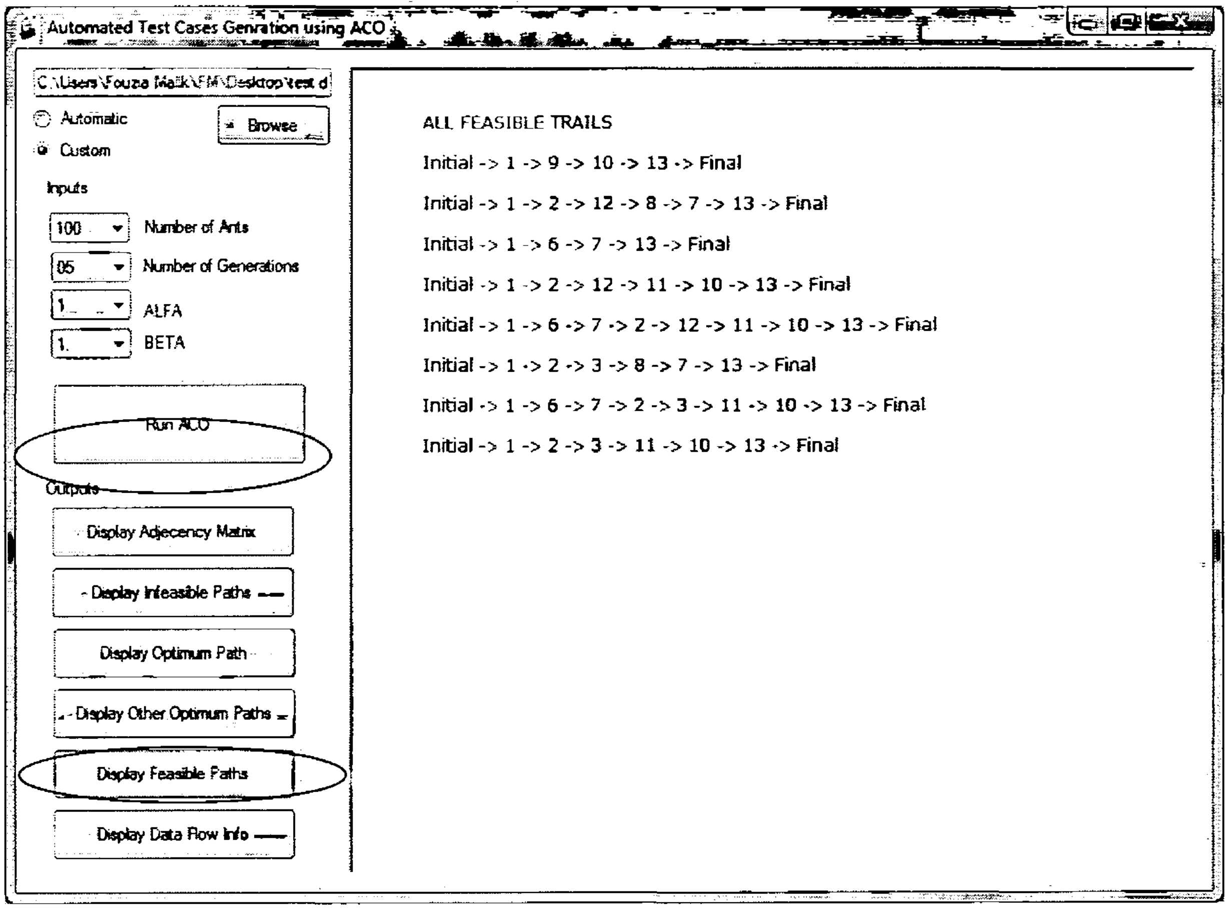
05 Number of Generations

1 ALFA

1 BETA

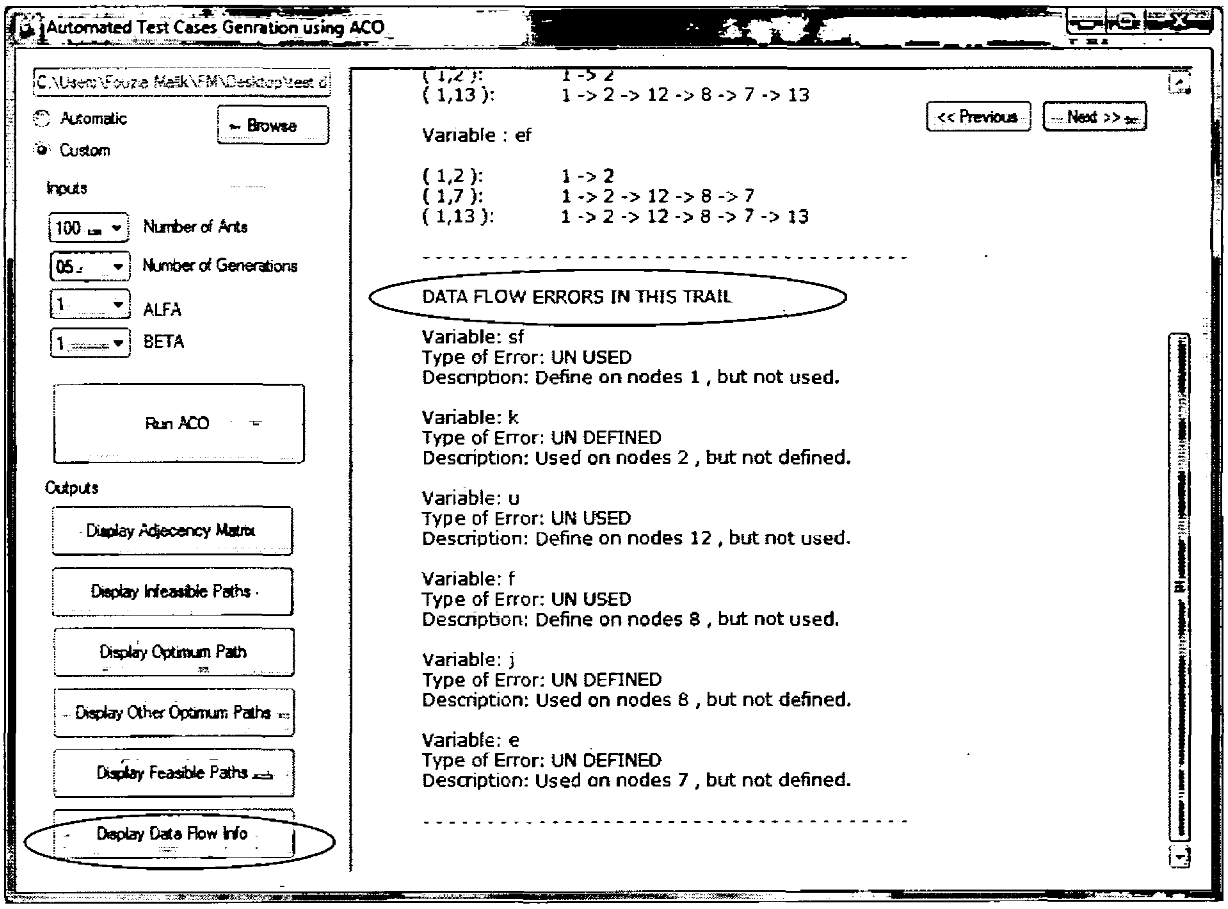
```
ADJECENCY MATRIX

0  1  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  1  0  0  0  1  0  0  1  0  0  0  0  0
0  0  1  1  0  0  0  0  0  0  0  0  1  0  0
0  0  0  0  1  1  0  0  1  0  0  1  0  0  0
0  0  0  0  0  1  0  0  0  0  0  0  0  0  0
0  0  0  1  0  1  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  1  0  0  0  0  0  0  0
0  0  1  0  0  0  0  0  0  0  0  0  0  1  0
0  0  0  0  0  0  0  1  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  1  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  1  0
0  0  0  0  0  0  0  0  0  0  1  0  0  0  0
0  0  0  0  0  0  0  0  1  0  0  1  0  0  0
0  0  1  1  0  0  0  0  0  0  0  0  0  0  1
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

D.2 Display Feasible Test Cases within ECS Model



```
ALL FEASIBLE TRAILS
Initial -> 1 -> 9 -> 10 -> 13 -> Final
Initial -> 1 -> 2 -> 12 -> 8 -> 7 -> 13 -> Final
Initial -> 1 -> 6 -> 7 -> 13 -> Final
Initial -> 1 -> 2 -> 12 -> 11 -> 10 -> 13 -> Final
Initial -> 1 -> 6 -> 7 -> 2 -> 12 -> 11 -> 10 -> 13 -> Final
Initial -> 1 -> 2 -> 3 -> 8 -> 7 -> 13 -> Final
Initial -> 1 -> 6 -> 7 -> 2 -> 3 -> 11 -> 10 -> 13 -> Final
Initial -> 1 -> 2 -> 3 -> 11 -> 10 -> 13 -> Final
```

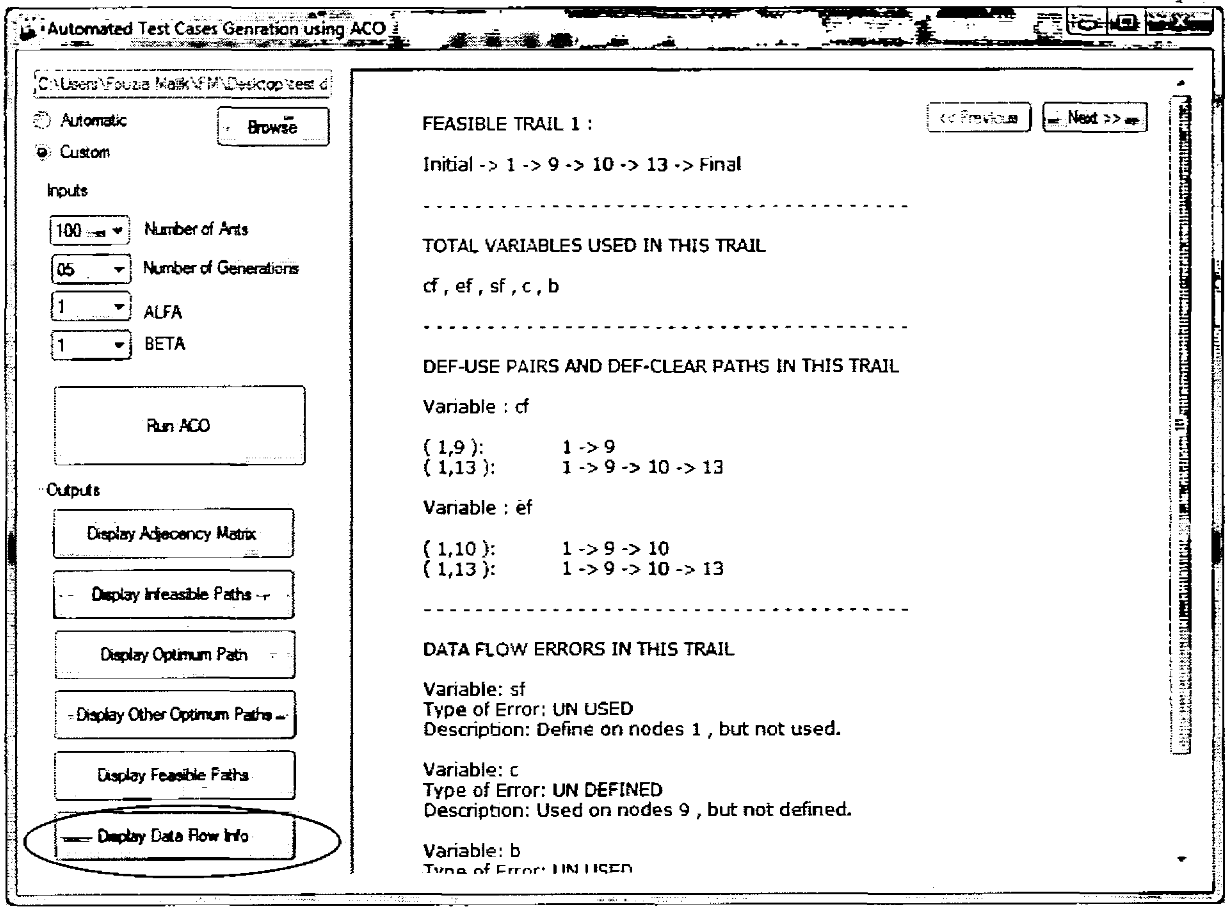
D.3 Display Data Flow Errors against Each Path of ECS Model



Inputs: Number of Ants: 100; Number of Generations: 05; ALFA: 1; BETA: 1

```
( 1,2 ):        1 -> 2
( 1,13 ):       1 -> 2 -> 12 -> 8 -> 7 -> 13

Variable : ef

( 1,2 ):        1 -> 2
( 1,7 ):        1 -> 2 -> 12 -> 8 -> 7
( 1,13 ):       1 -> 2 -> 12 -> 8 -> 7 -> 13

-----------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: sf
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: k
Type of Error: UN DEFINED
Description: Used on nodes 2 , but not defined.

Variable: u
Type of Error: UN USED
Description: Define on nodes 12 , but not used.

Variable: f
Type of Error: UN USED
Description: Define on nodes 8 , but not used.

Variable: j
Type of Error: UN DEFINED
Description: Used on nodes 8 , but not defined.

Variable: e
Type of Error: UN DEFINED
Description: Used on nodes 7 , but not defined.

-----------------------------------
```

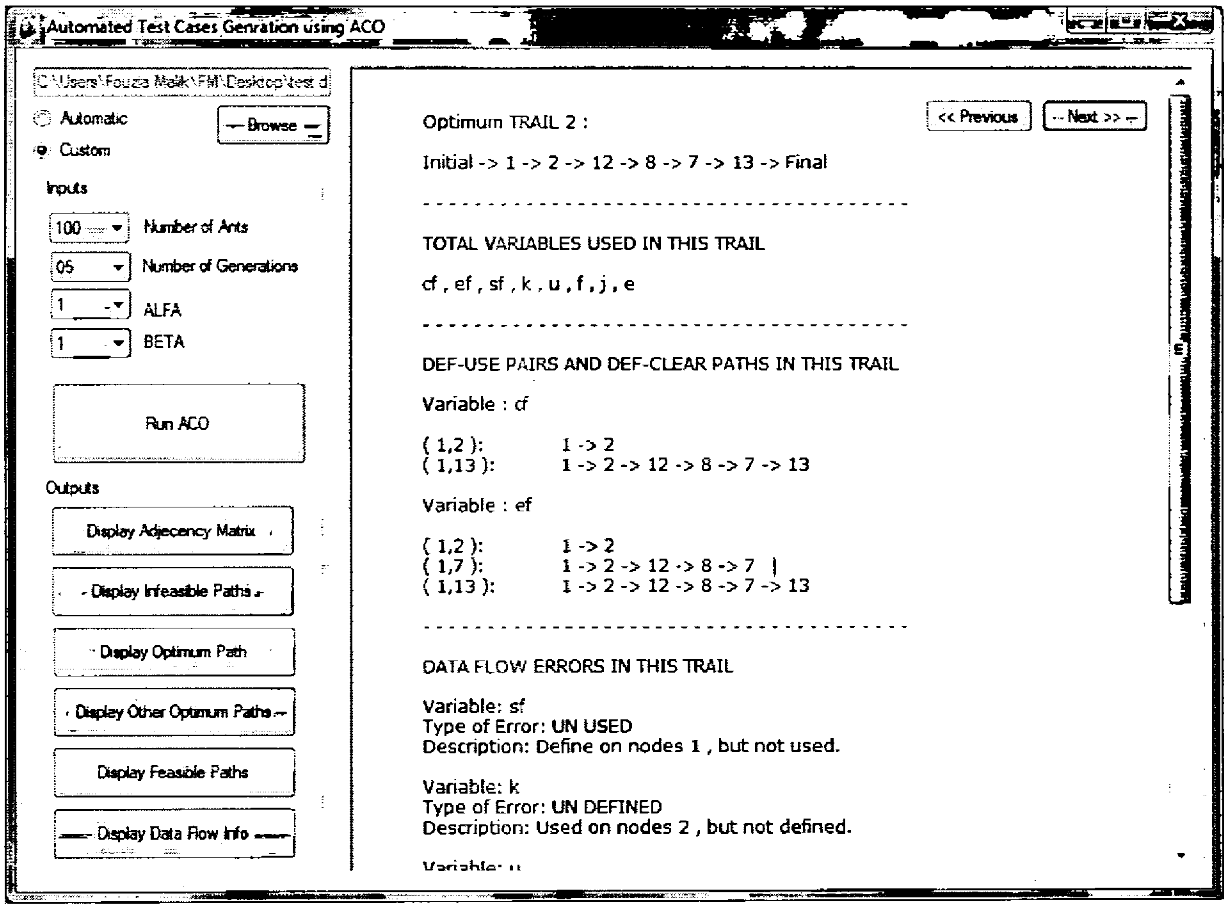
D.4 Display Def-Use Pairs, All Def-use Paths & Data Flow Errors of ECS Model



Number of Ants: 100
Number of Generations: 05
ALFA: 1
BETA: 1

```
FEASIBLE TRAIL 1 :

Initial -> 1 -> 9 -> 10 -> 13 -> Final

-----------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

cf , ef , sf , c , b

-----------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : cf

( 1,9 ):        1 -> 9
( 1,13 ):       1 -> 9 -> 10 -> 13

Variable : ef

( 1,10 ):       1 -> 9 -> 10
( 1,13 ):       1 -> 9 -> 10 -> 13

-----------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: sf
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: c
Type of Error: UN DEFINED
Description: Used on nodes 9 , but not defined.

Variable: b
```

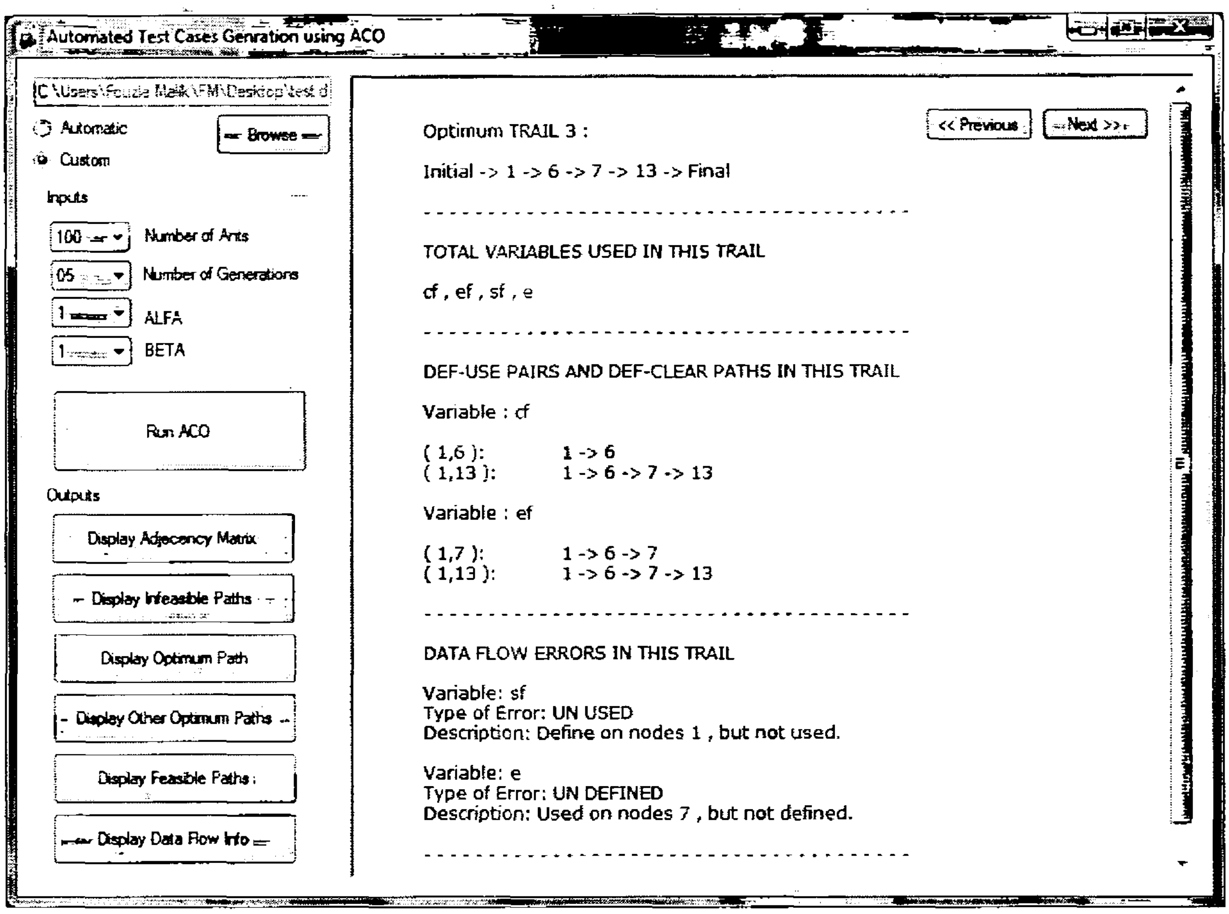
All Def-Use Pairs, Def-Use paths & Data Flow Errors of Feasible Test Case 2 of ECS Model



Number of Ants: 100
Number of Generations: 05
ALFA: 1
BETA: 1

```
Optimum TRAIL 2 :

Initial -> 1 -> 2 -> 12 -> 8 -> 7 -> 13 -> Final

------------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

cf , ef , sf , k , u , f , j , e

------------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : cf

( 1,2 ):        1 -> 2
( 1,13 ):       1 -> 2 -> 12 -> 8 -> 7 -> 13

Variable : ef

( 1,2 ):        1 -> 2
( 1,7 ):        1 -> 2 -> 12 -> 8 -> 7
( 1,13 ):       1 -> 2 -> 12 -> 8 -> 7 -> 13

------------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: sf
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: k
Type of Error: UN DEFINED
Description: Used on nodes 2 , but not defined.
```

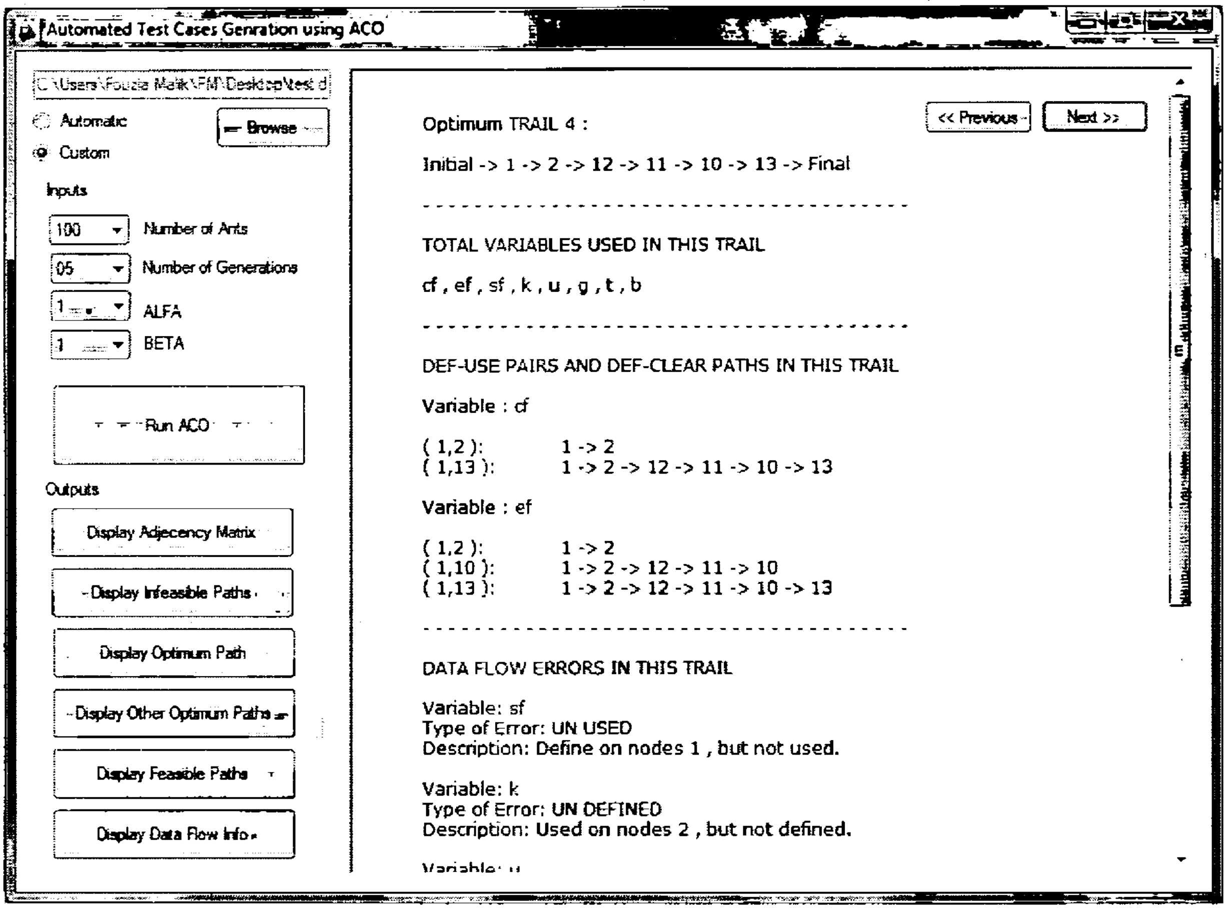
All Def-Use Pairs, Def-Use paths & Data Flow Errors of Feasible Test Case 3 of ECS Model



Inputs: Number of Ants 100; Number of Generations 05; ALFA 1; BETA 1

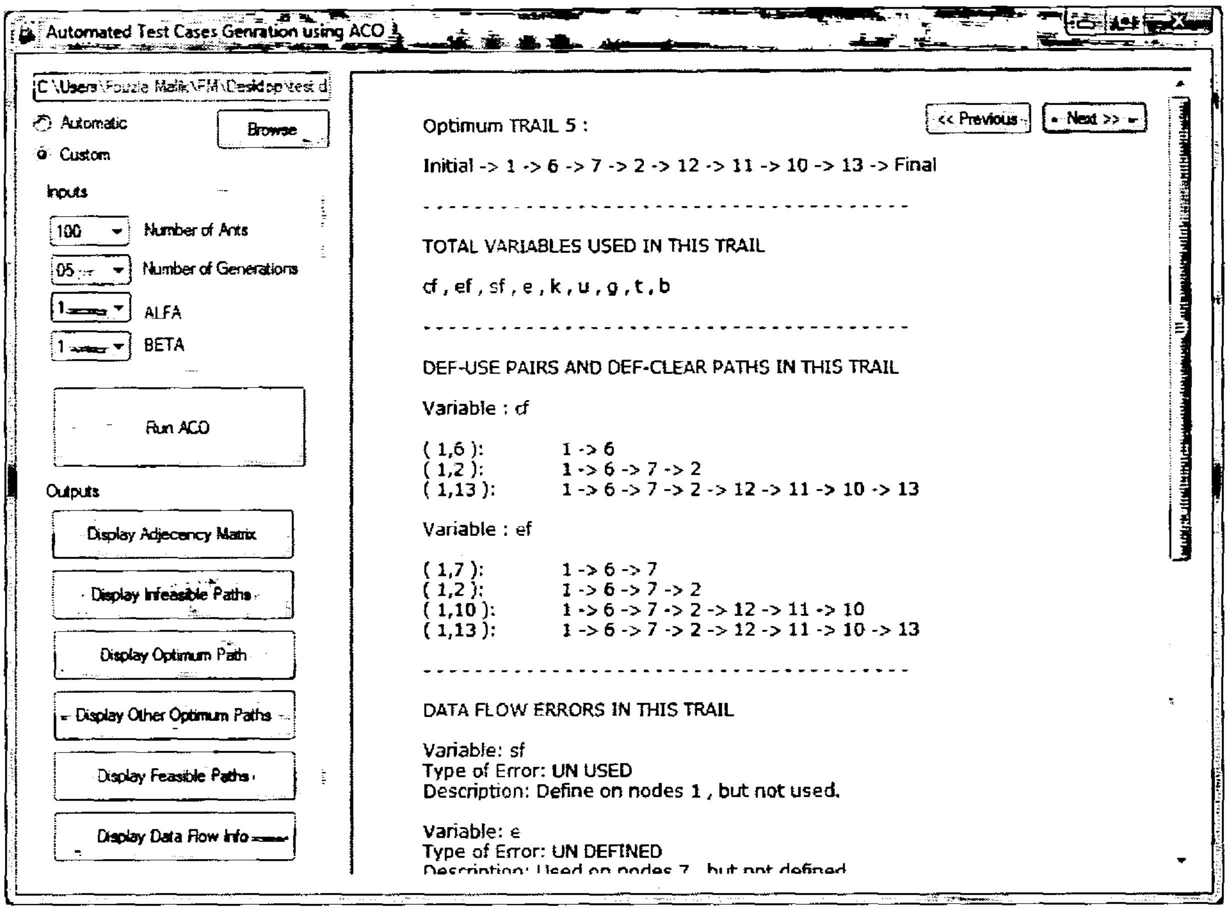
```
Optimum TRAIL 3 :

Initial -> 1 -> 6 -> 7 -> 13 -> Final

------------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

cf , ef , sf , e

------------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : cf

( 1,6 ):        1 -> 6
( 1,13 ):       1 -> 6 -> 7 -> 13

Variable : ef

( 1,7 ):        1 -> 6 -> 7
( 1,13 ):       1 -> 6 -> 7 -> 13

------------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: sf
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: e
Type of Error: UN DEFINED
Description: Used on nodes 7 , but not defined.

------------------------------------
```

All Def-Use Pairs, Def-Use paths & Data Flow Errors of Feasible Test Case 4 of ECS Model



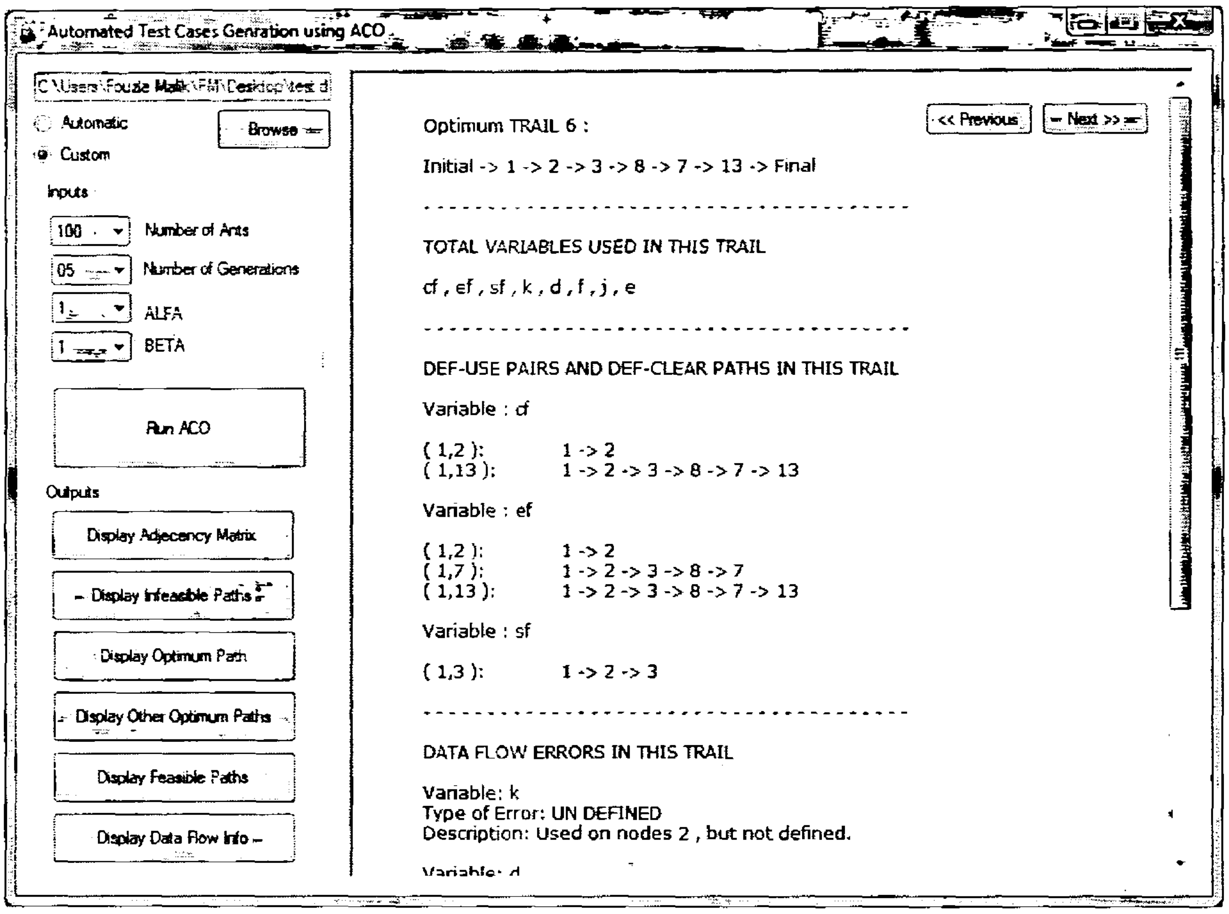
```
Optimum TRAIL 4 :

Initial -> 1 -> 2 -> 12 -> 11 -> 10 -> 13 -> Final

------------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

cf , ef , sf , k , u , g , t , b

------------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : cf

( 1,2 ):        1 -> 2
( 1,13 ):       1 -> 2 -> 12 -> 11 -> 10 -> 13

Variable : ef

( 1,2 ):        1 -> 2
( 1,10 ):       1 -> 2 -> 12 -> 11 -> 10
( 1,13 ):       1 -> 2 -> 12 -> 11 -> 10 -> 13

------------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: sf
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: k
Type of Error: UN DEFINED
Description: Used on nodes 2 , but not defined.

Variable: u
```

All Def-Use Pairs, Def-Use paths & Data Flow Errors of Feasible Test Case 5 of ECS Model



```
Optimum TRAIL 5 :
Initial -> 1 -> 6 -> 7 -> 2 -> 12 -> 11 -> 10 -> 13 -> Final
-----------------------------------
TOTAL VARIABLES USED IN THIS TRAIL
cf , ef , sf , e , k , u , g , t , b
-----------------------------------
DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL
Variable : cf
( 1,6 ):        1 -> 6
( 1,2 ):        1 -> 6 -> 7 -> 2
( 1,13 ):       1 -> 6 -> 7 -> 2 -> 12 -> 11 -> 10 -> 13
Variable : ef
( 1,7 ):        1 -> 6 -> 7
( 1,2 ):        1 -> 6 -> 7 -> 2
( 1,10 ):       1 -> 6 -> 7 -> 2 -> 12 -> 11 -> 10
( 1,13 ):       1 -> 6 -> 7 -> 2 -> 12 -> 11 -> 10 -> 13
-----------------------------------
DATA FLOW ERRORS IN THIS TRAIL
Variable: sf
Type of Error: UN USED
Description: Define on nodes 1 , but not used.
Variable: e
Type of Error: UN DEFINED
Description: Used on nodes 7 , but not defined
```

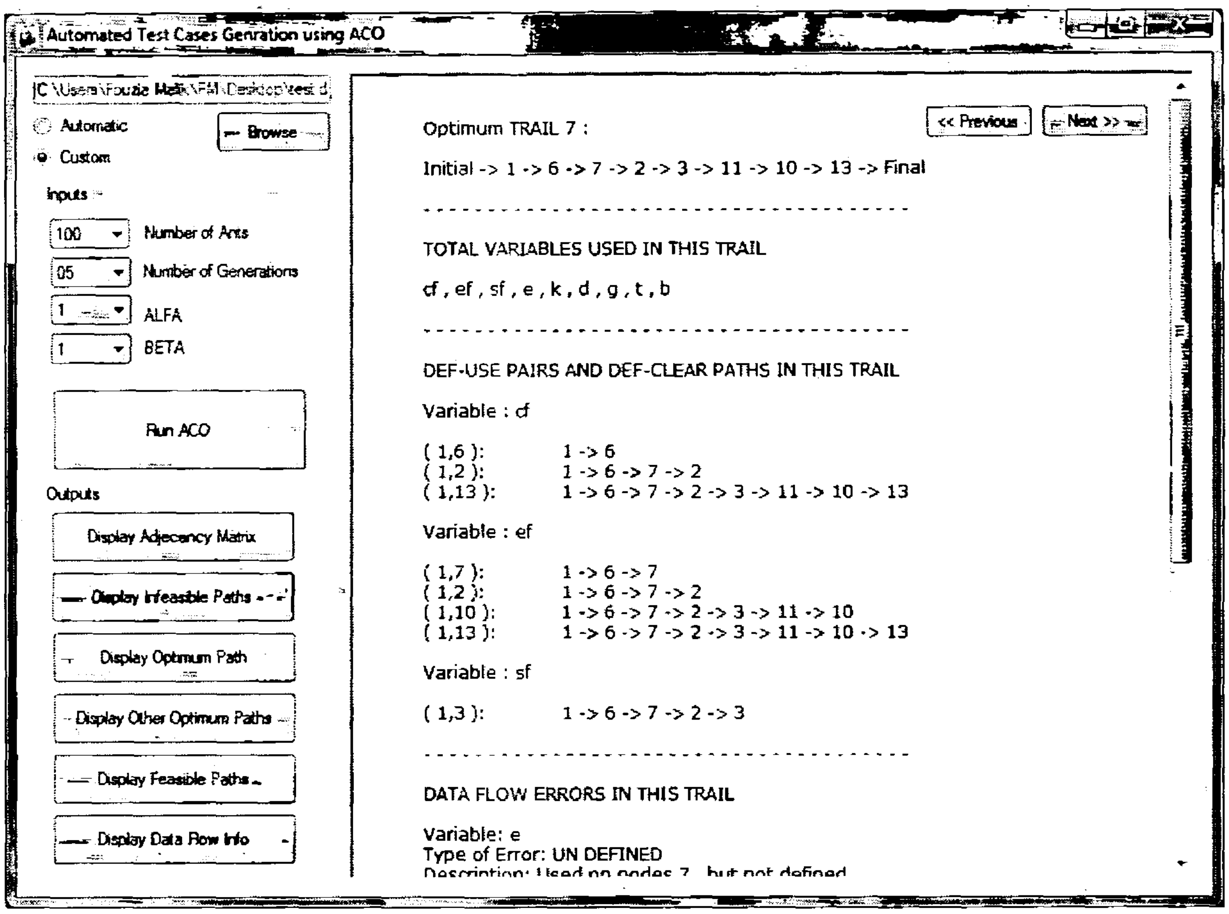
All Def-Use Pairs, Def-Use paths & Data Flow Errors of Feasible Test Case 6 of ECS Model



Number of Ants: 100
Number of Generations: 05
ALFA: 1
BETA: 1

```
Optimum TRAIL 6 :

Initial -> 1 -> 2 -> 3 -> 8 -> 7 -> 13 -> Final

-----------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

cf , ef , sf , k , d , f , j , e

-----------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : cf

( 1,2 ):        1 -> 2
( 1,13 ):       1 -> 2 -> 3 -> 8 -> 7 -> 13

Variable : ef

( 1,2 ):        1 -> 2
( 1,7 ):        1 -> 2 -> 3 -> 8 -> 7
( 1,13 ):       1 -> 2 -> 3 -> 8 -> 7 -> 13

Variable : sf

( 1,3 ):        1 -> 2 -> 3

-----------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: k
Type of Error: UN DEFINED
Description: Used on nodes 2 , but not defined.

Variable: d
```

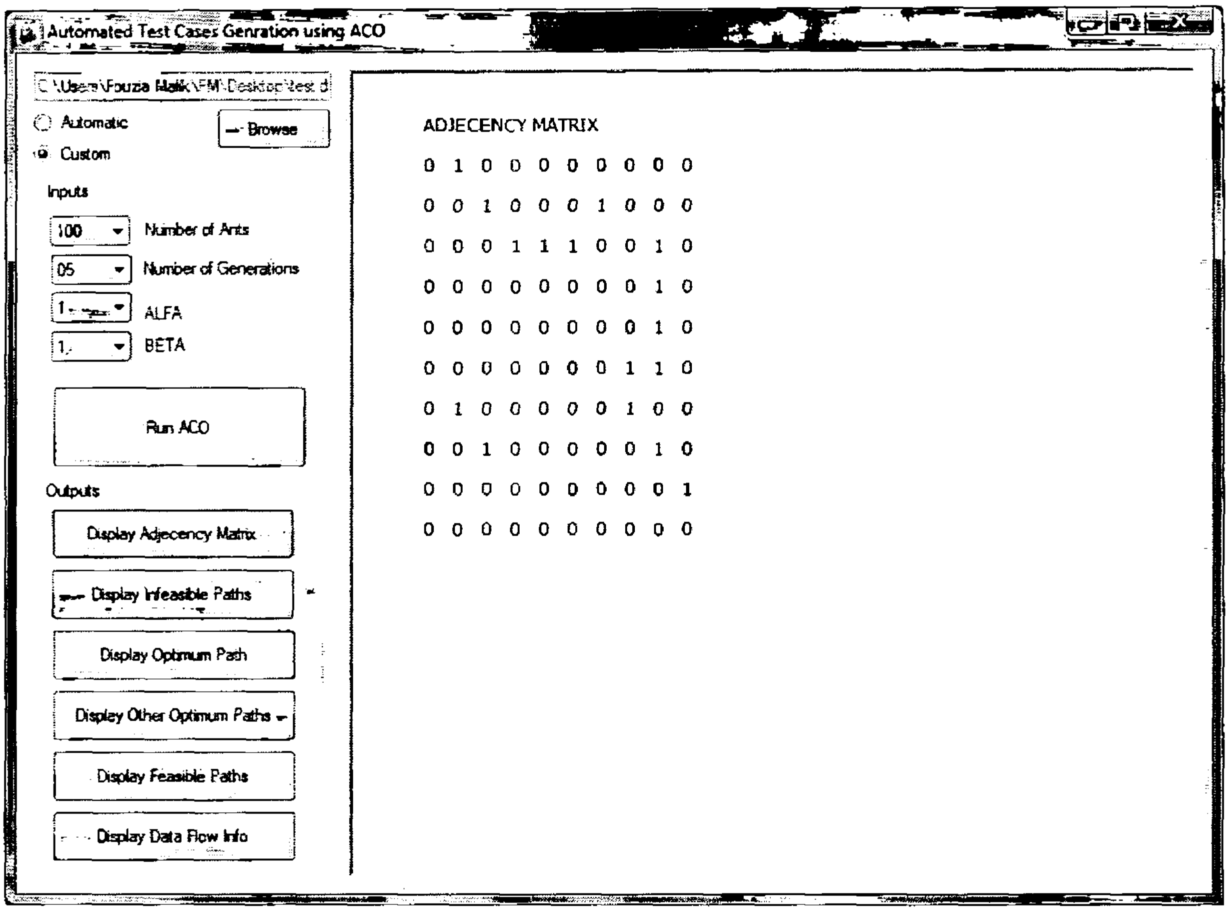
All Def-Use Pairs, Def-Use paths & Data Flow Errors of Feasible Test Case 7 of ECS Model



Number of Ants: 100
Number of Generations: 05
ALFA: 1
BETA: 1

```
Optimum TRAIL 7 :

Initial -> 1 -> 6 -> 7 -> 2 -> 3 -> 11 -> 10 -> 13 -> Final

-----------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

cf , ef , sf , e , k , d , g , t , b

-----------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : cf

( 1,6 ):        1 -> 6
( 1,2 ):        1 -> 6 -> 7 -> 2
( 1,13 ):       1 -> 6 -> 7 -> 2 -> 3 -> 11 -> 10 -> 13

Variable : ef

( 1,7 ):        1 -> 6 -> 7
( 1,2 ):        1 -> 6 -> 7 -> 2
( 1,10 ):       1 -> 6 -> 7 -> 2 -> 3 -> 11 -> 10
( 1,13 ):       1 -> 6 -> 7 -> 2 -> 3 -> 11 -> 10 -> 13

Variable : sf

( 1,3 ):        1 -> 6 -> 7 -> 2 -> 3

-----------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: e
Type of Error: UN DEFINED
Description: Used on nodes 7 , but not defined
```

D.5 Display Adjacency Matrix of Telephone System Model



C:\Users\Fouzia Malik\FM\Desktop\Test d

Automatic [✓ Custom]

100 Number of Ants

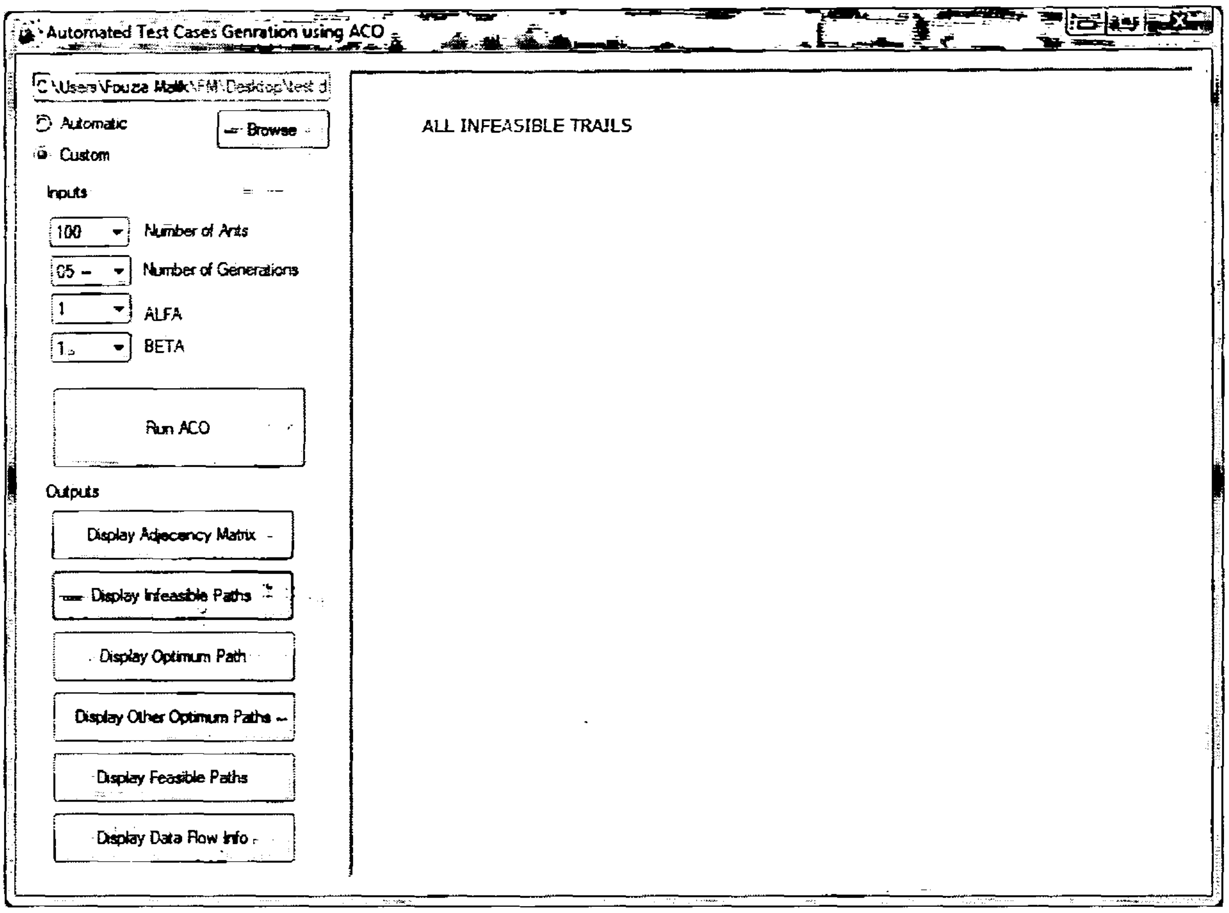
05 Number of Generations

1 ALFA

1 BETA

```
ADJECENCY MATRIX
0 1 0 0 0 0 0 0 0 0
0 0 1 0 0 0 1 0 0 0
0 0 0 1 1 1 0 0 1 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1 1 0
0 1 0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0
```

D.6 Display Infeasible paths within Telephone System Model



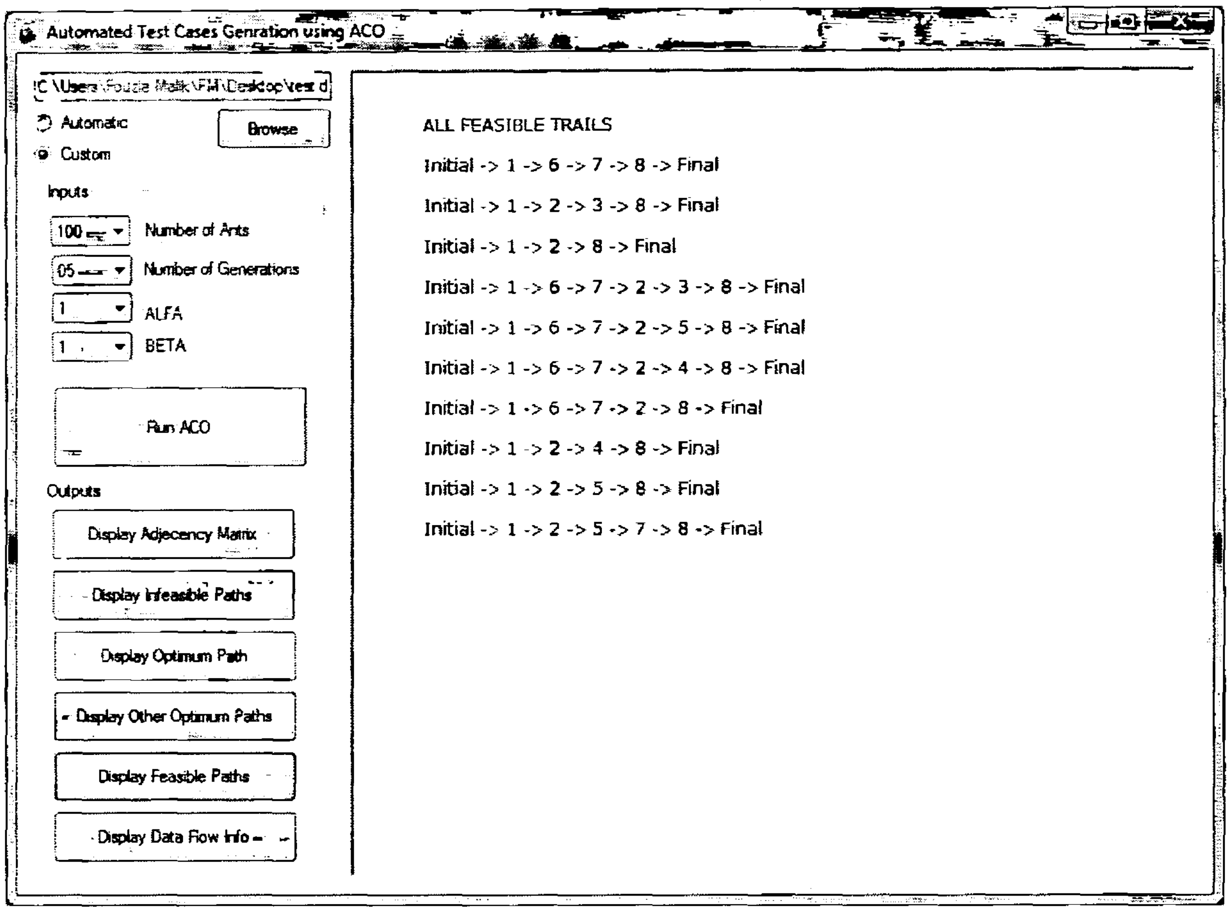
C:\Users\Fouza Malik\FM\Desktop\test d

Automatic
[✓ Custom]

100 Number of Ants
05 Number of Generations
1 ALFA
1 BETA

ALL INFEASIBLE TRAILS

D.7 Display Feasible Test Cases within Telephone System Model

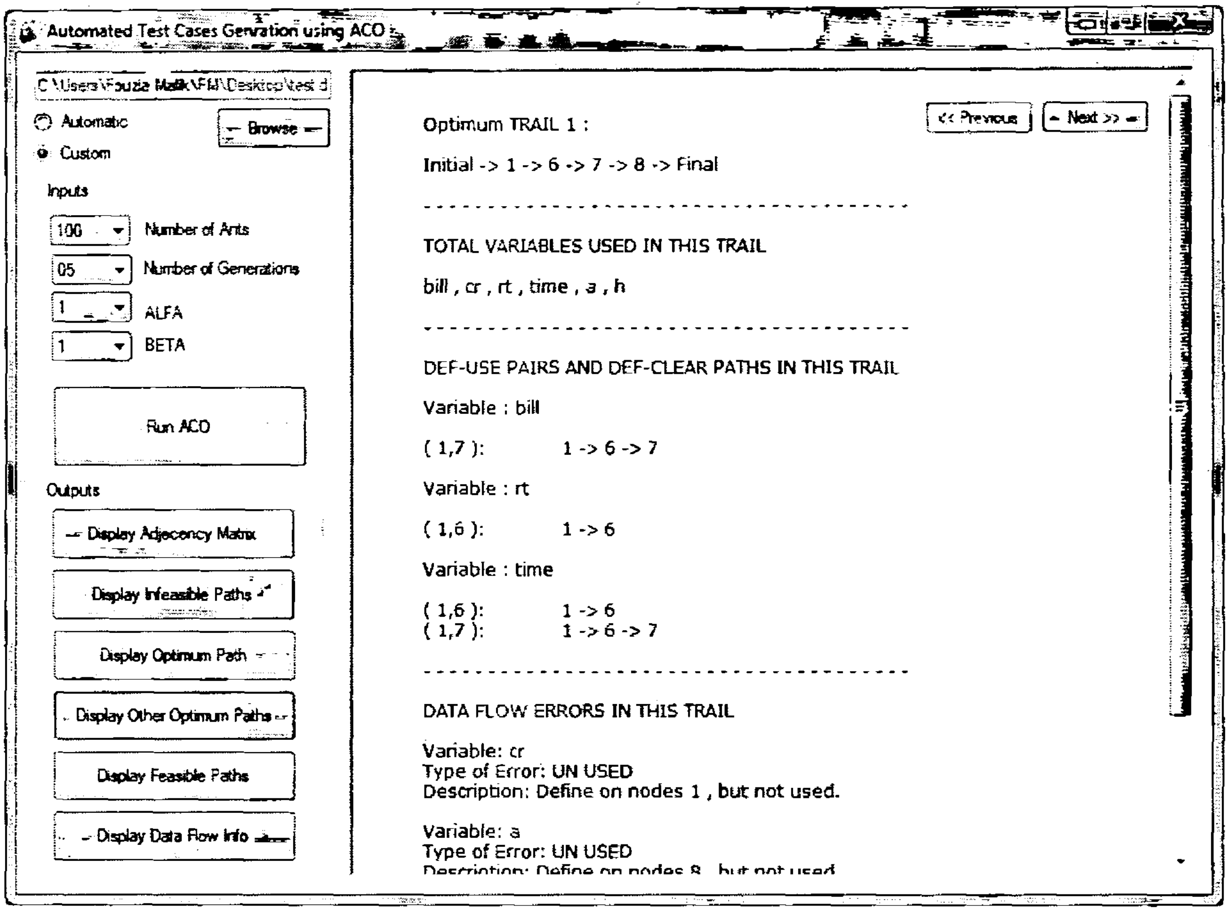


Custom [✓]

100 Number of Ants
05 Number of Generations
1 ALFA
1 BETA

```
ALL FEASIBLE TRAILS
Initial -> 1 -> 6 -> 7 -> 8 -> Final
Initial -> 1 -> 2 -> 3 -> 8 -> Final
Initial -> 1 -> 2 -> 8 -> Final
Initial -> 1 -> 6 -> 7 -> 2 -> 3 -> 8 -> Final
Initial -> 1 -> 6 -> 7 -> 2 -> 5 -> 8 -> Final
Initial -> 1 -> 6 -> 7 -> 2 -> 4 -> 8 -> Final
Initial -> 1 -> 6 -> 7 -> 2 -> 8 -> Final
Initial -> 1 -> 2 -> 4 -> 8 -> Final
Initial -> 1 -> 2 -> 5 -> 8 -> Final
Initial -> 1 -> 2 -> 5 -> 7 -> 8 -> Final
```

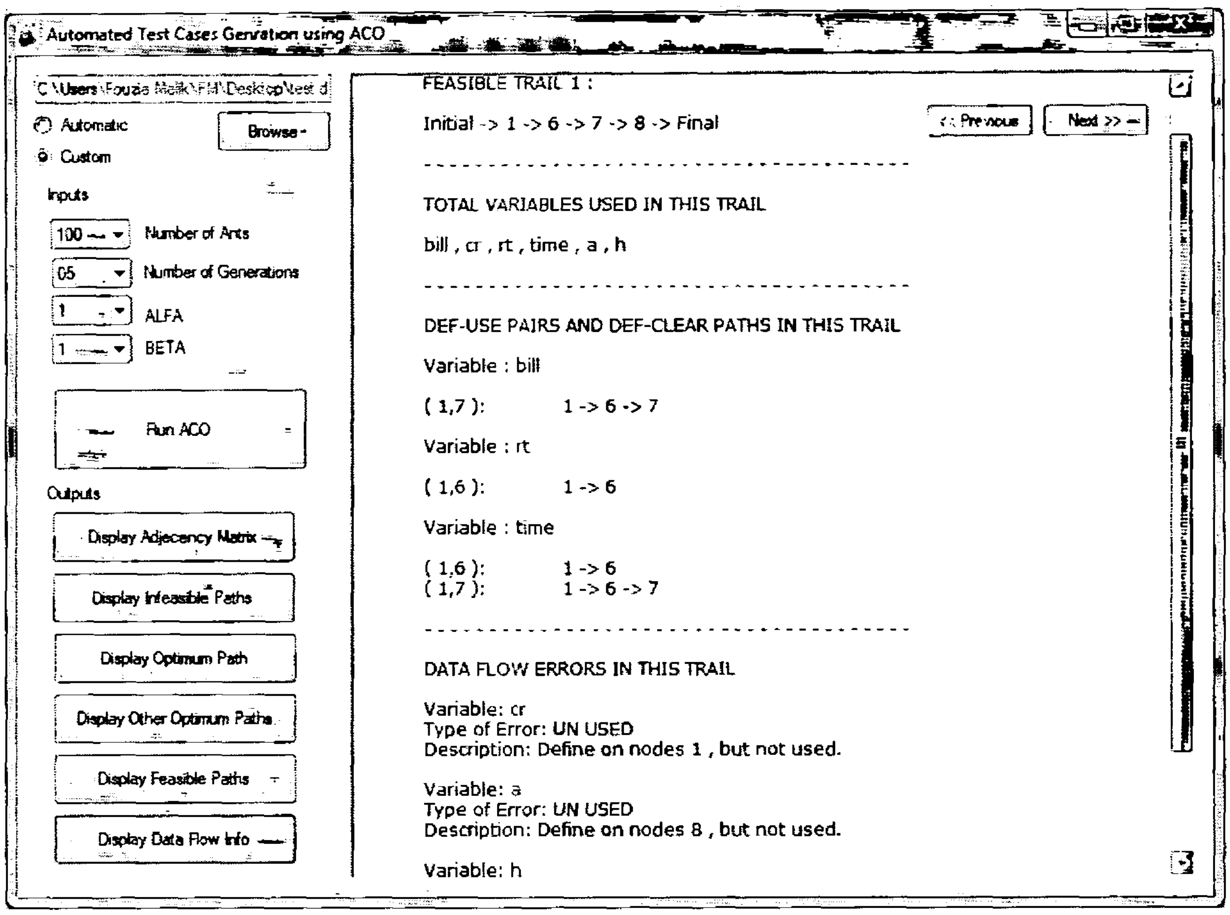
D.8 Display Optimum Paths within Telephone System Model



Number of Ants: 100
Number of Generations: 05
ALFA: 1
BETA: 1

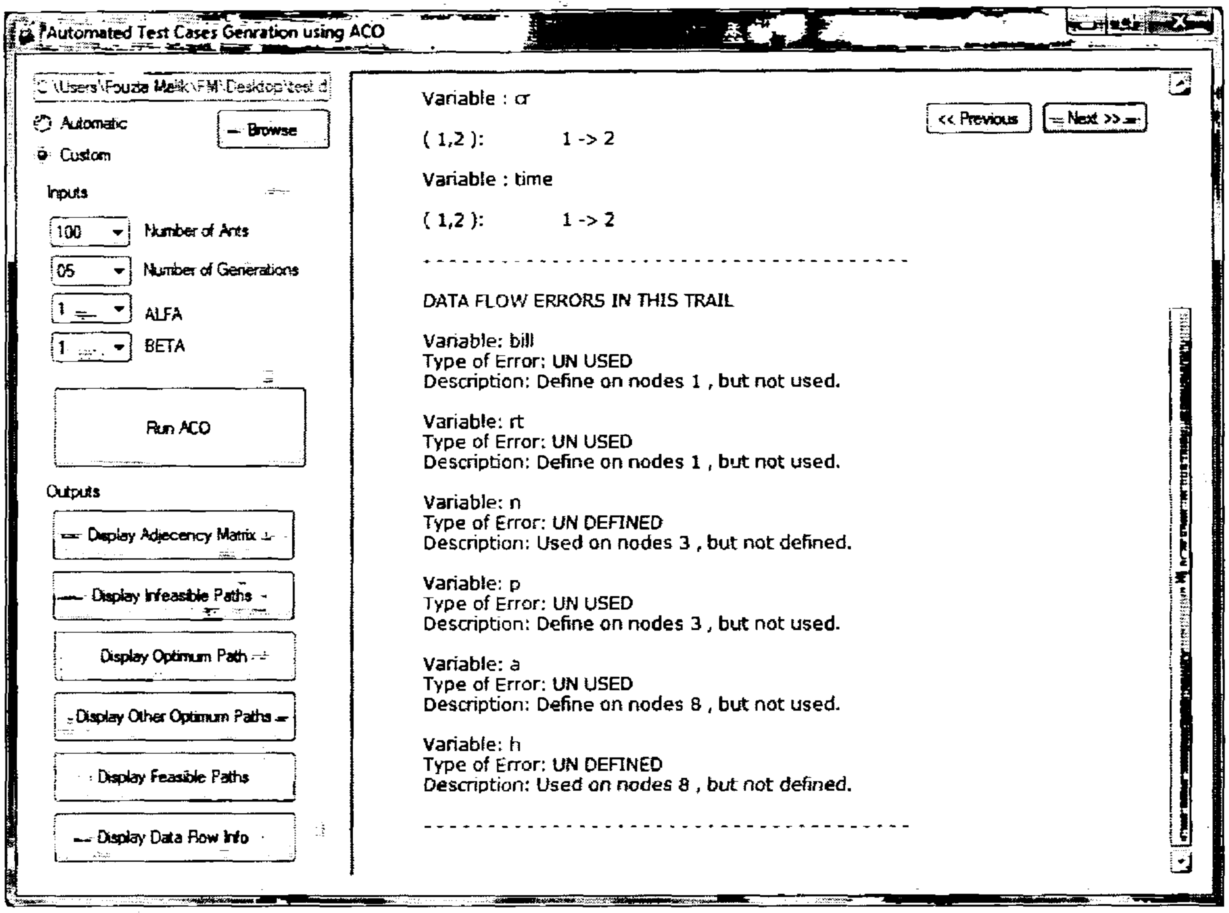
```
Optimum TRAIL 1 :

Initial -> 1 -> 6 -> 7 -> 8 -> Final

-------------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

bill , cr , rt , time , a , h

-------------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : bill

( 1,7 ):        1 -> 6 -> 7

Variable : rt

( 1,6 ):        1 -> 6

Variable : time

( 1,6 ):        1 -> 6
( 1,7 ):        1 -> 6 -> 7

-------------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: cr
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: a
Type of Error: UN USED
Description: Define on nodes 8 , but not used
```

D.9 Display All Def-Use Pairs, All Def-Use Paths against Each Path within Telephone System Model



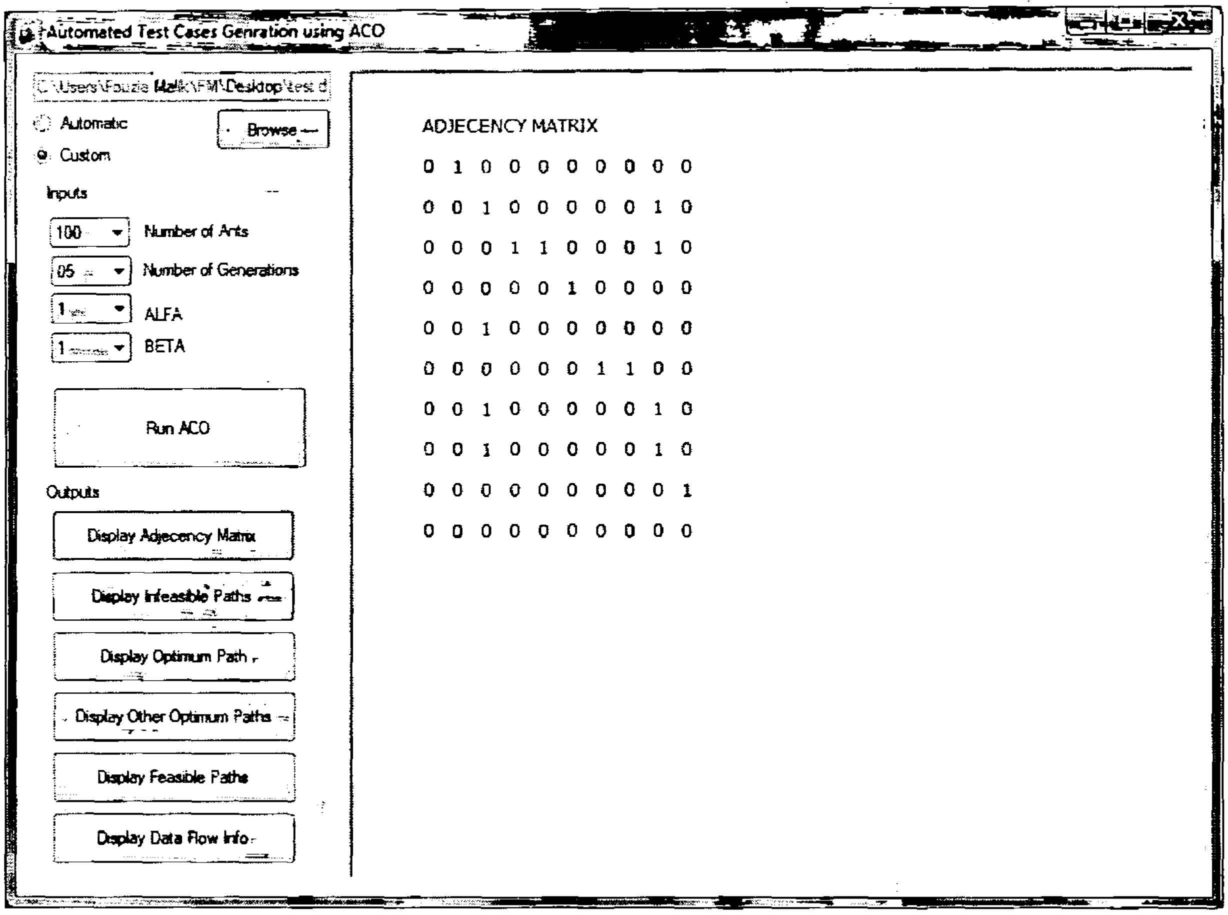
```
FEASIBLE TRAIL 1 :

Initial -> 1 -> 6 -> 7 -> 8 -> Final

-------------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

bill , cr , rt , time , a , h

-------------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : bill

( 1,7 ):        1 -> 6 -> 7

Variable : rt

( 1,6 ):        1 -> 6

Variable : time

( 1,6 ):        1 -> 6
( 1,7 ):        1 -> 6 -> 7

-------------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: cr
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: a
Type of Error: UN USED
Description: Define on nodes 8 , but not used.

Variable: h
```

D.10 Display Data Flow Errors against Each Path within Telephone System Model



```
Variable : cr

( 1,2 ):          1 -> 2

Variable : time

( 1,2 ):          1 -> 2

-------------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: bill
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: rt
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: n
Type of Error: UN DEFINED
Description: Used on nodes 3 , but not defined.

Variable: p
Type of Error: UN USED
Description: Define on nodes 3 , but not used.

Variable: a
Type of Error: UN USED
Description: Define on nodes 8 , but not used.

Variable: h
Type of Error: UN DEFINED
Description: Used on nodes 8 , but not defined.

-------------------------------------
```

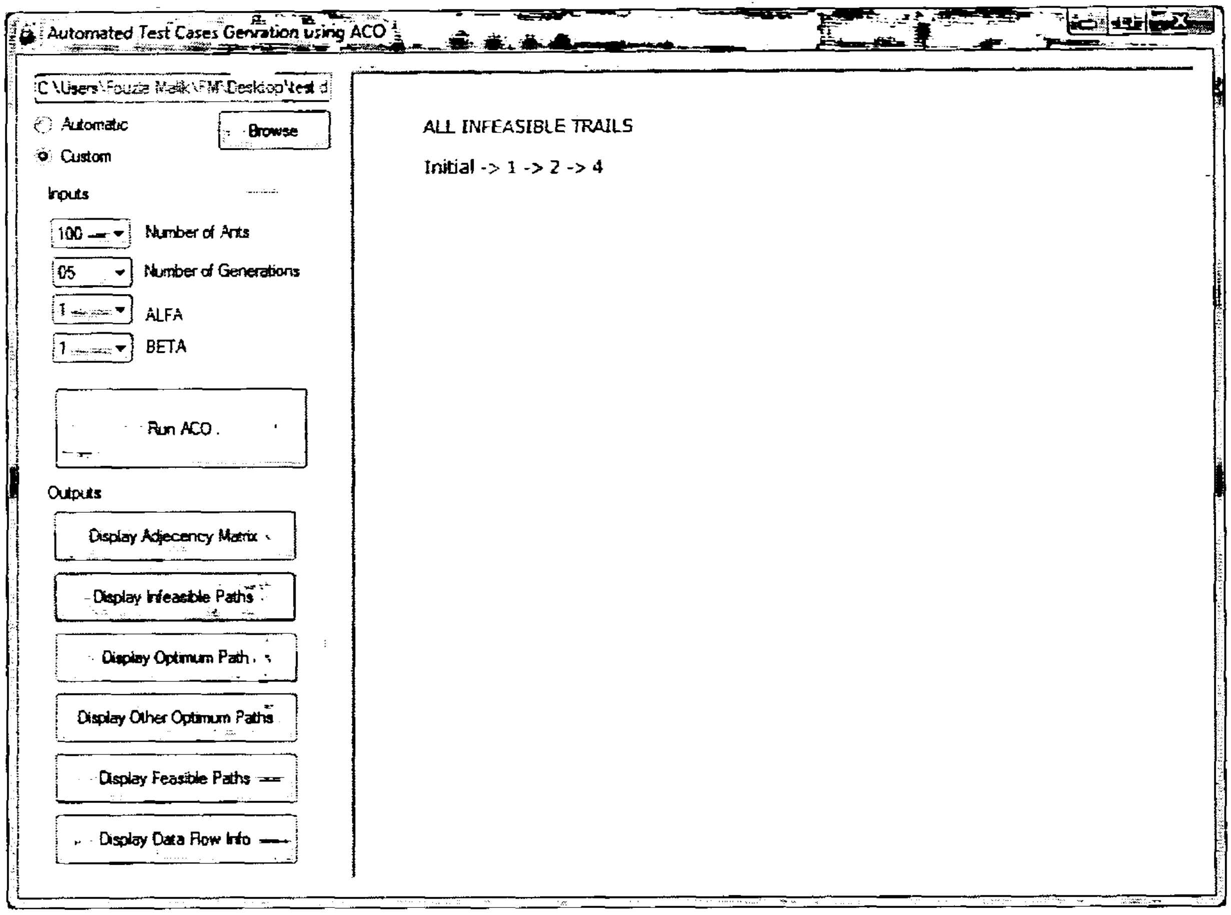
D.11 Display Adjacency Matrix of Library Management System Model



Number of Ants: 100
Number of Generations: 05
ALFA: 1
BETA: 1

```
ADJECENCY MATRIX
0 1 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 1 0
0 0 0 1 1 0 0 0 1 0
0 0 0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 0 0
0 0 1 0 0 0 0 0 1 0
0 0 1 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0
```

D.12 Display Infeasible Paths within LMS Model



C:\Users\Fouzia Malik\FM\Desktop\test.d

( ) Automatic
(•) Custom

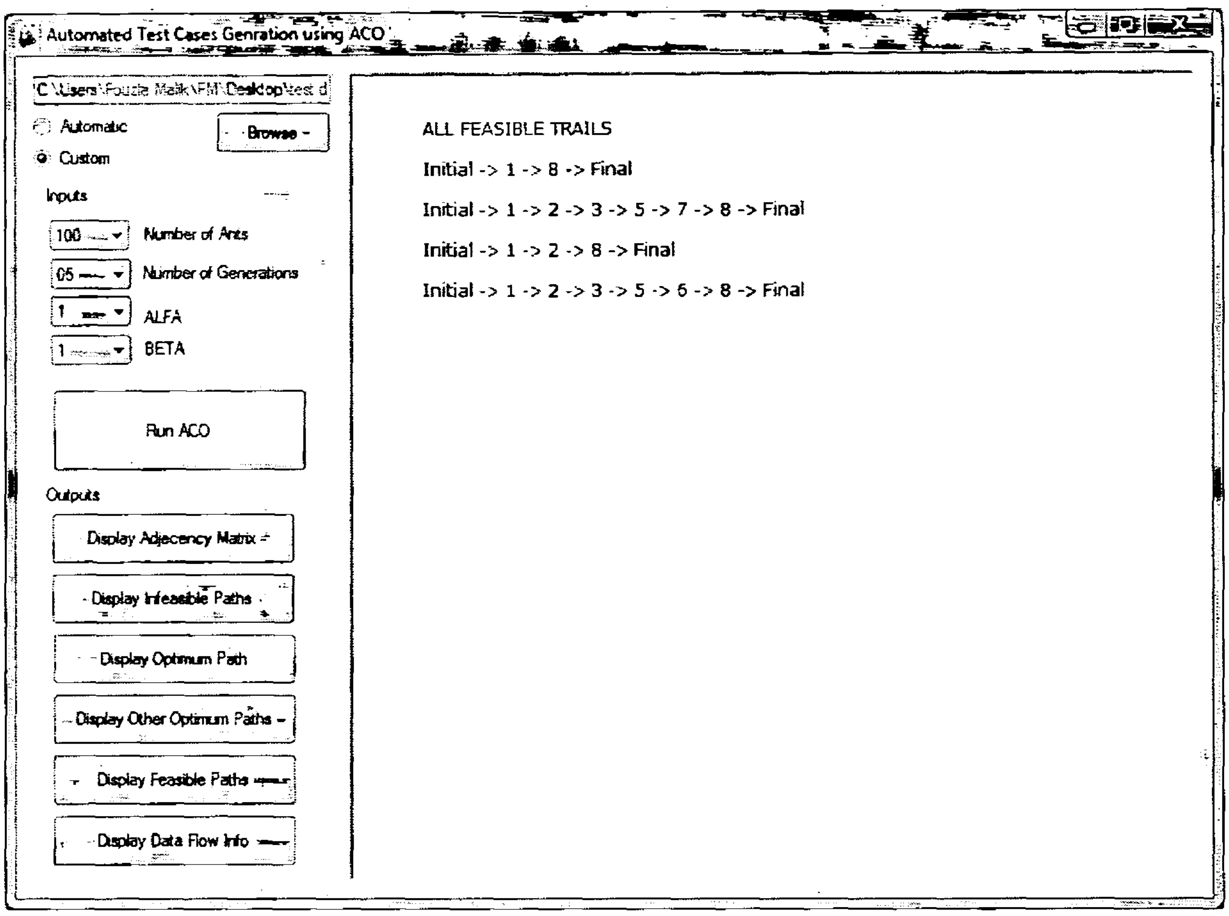
Inputs

100 Number of Ants
05 Number of Generations
1 ALFA
1 BETA

Outputs

```
ALL INFEASIBLE TRAILS
Initial -> 1 -> 2 -> 4
```

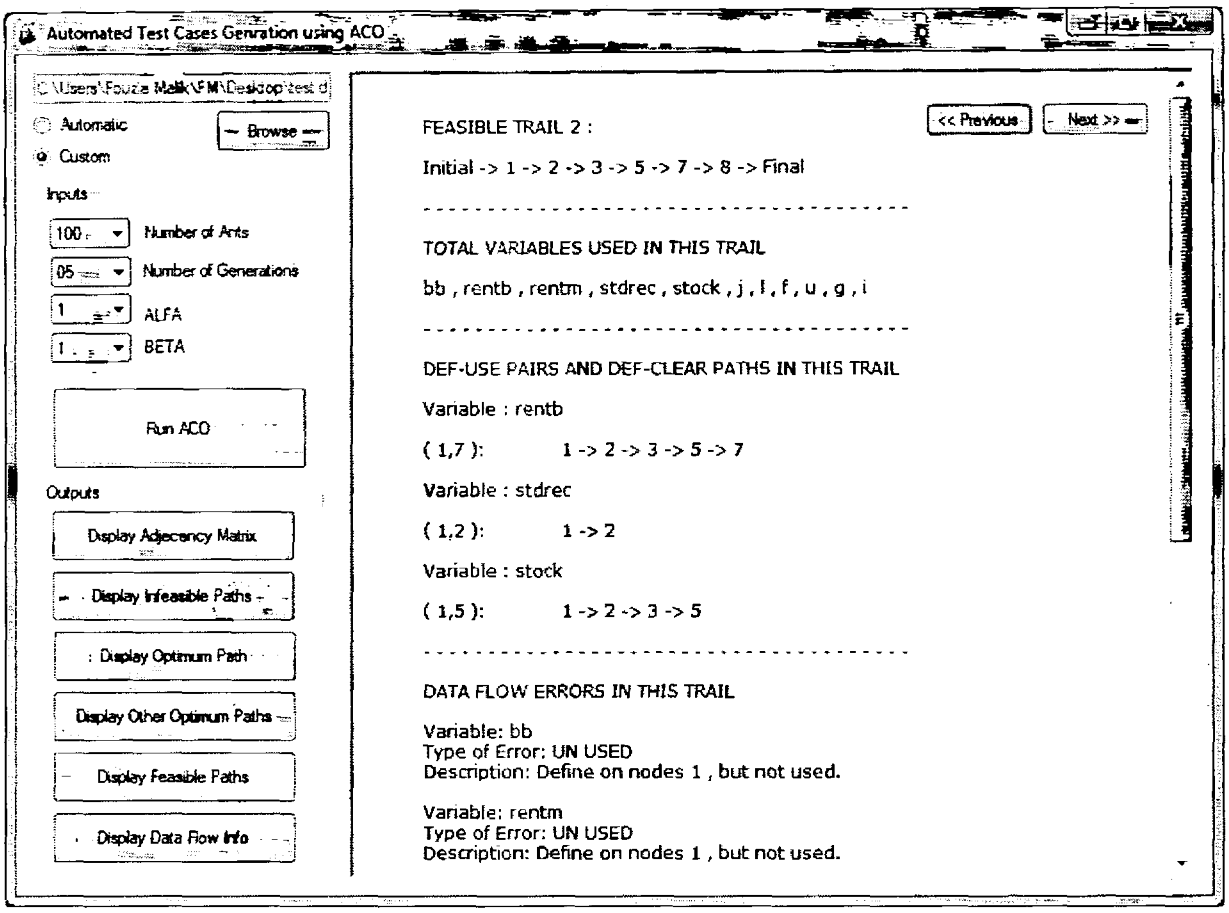
D.13 Display Feasible Test Cases within LMS Model



Number of Ants: 100
Number of Generations: 05
ALFA: 1
BETA: 1

```
ALL FEASIBLE TRAILS
Initial -> 1 -> 8 -> Final
Initial -> 1 -> 2 -> 3 -> 5 -> 7 -> 8 -> Final
Initial -> 1 -> 2 -> 8 -> Final
Initial -> 1 -> 2 -> 3 -> 5 -> 6 -> 8 -> Final
```

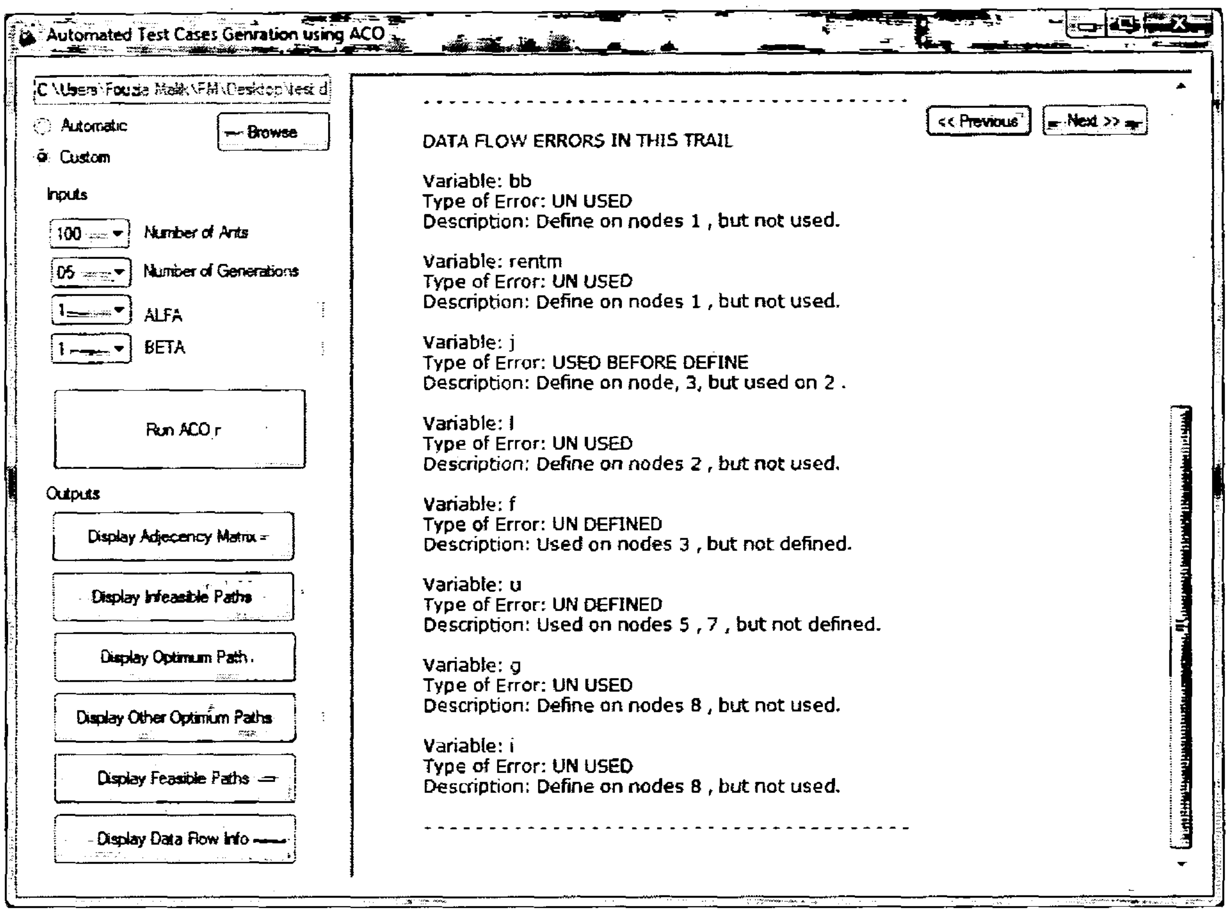
D.14 Display All Def-Use Pairs & All Def-Use Paths within LMS Model



Number of Ants: 100
Number of Generations: 05
ALFA: 1
BETA: 1

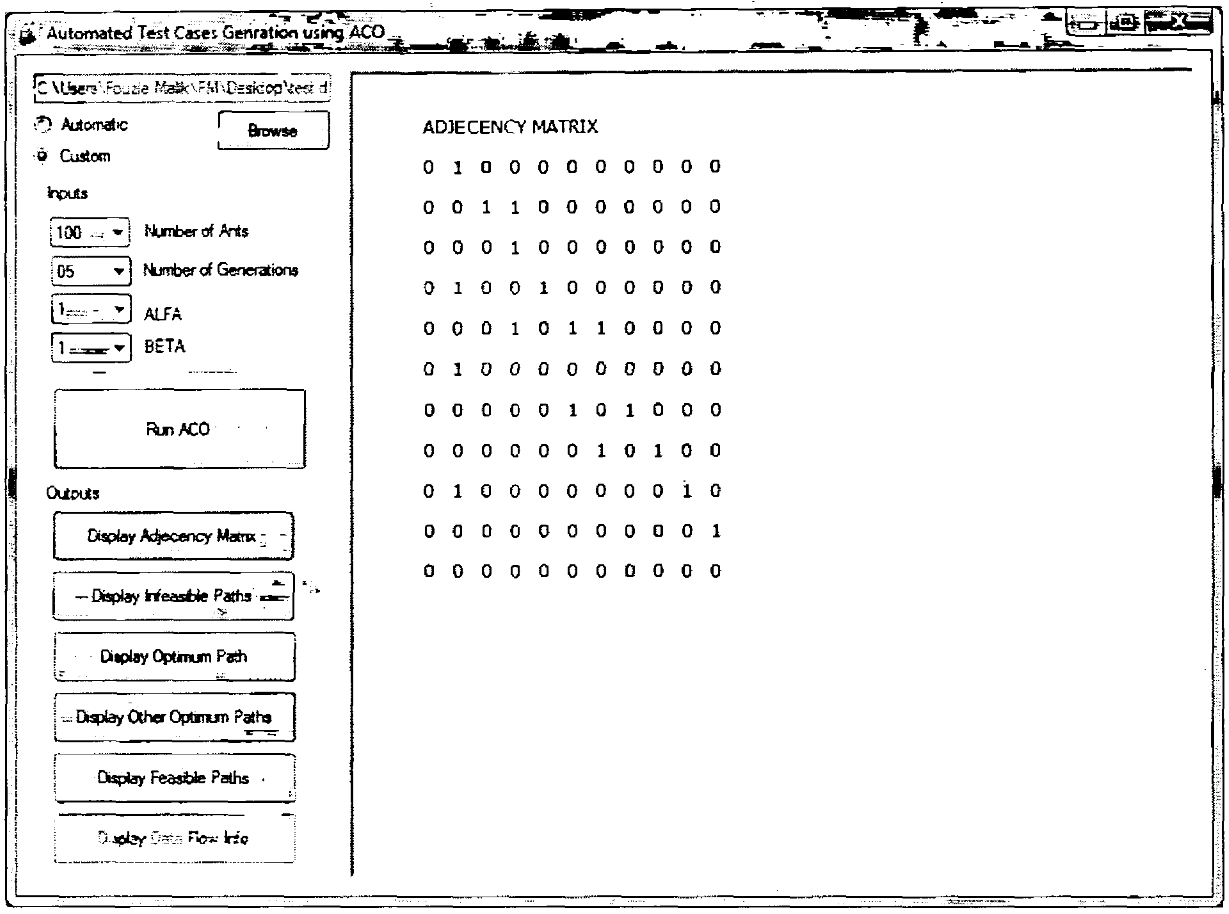
```
FEASIBLE TRAIL 2 :

Initial -> 1 -> 2 -> 3 -> 5 -> 7 -> 8 -> Final

-----------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

bb , rentb , rentm , stdrec , stock , j , l , f , u , g , i

-----------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : rentb

( 1,7 ):        1 -> 2 -> 3 -> 5 -> 7

Variable : stdrec

( 1,2 ):        1 -> 2

Variable : stock

( 1,5 ):        1 -> 2 -> 3 -> 5

-----------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: bb
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: rentm
Type of Error: UN USED
Description: Define on nodes 1 , but not used.
```

D.15 Display Data Flow Errors within LMS Model



```
------------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: bb
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: rentm
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: j
Type of Error: USED BEFORE DEFINE
Description: Define on node, 3, but used on 2 .

Variable: l
Type of Error: UN USED
Description: Define on nodes 2 , but not used.

Variable: f
Type of Error: UN DEFINED
Description: Used on nodes 3 , but not defined.

Variable: u
Type of Error: UN DEFINED
Description: Used on nodes 5 , 7 , but not defined.

Variable: g
Type of Error: UN USED
Description: Define on nodes 8 , but not used.

Variable: i
Type of Error: UN USED
Description: Define on nodes 8 , but not used.

------------------------------------
```

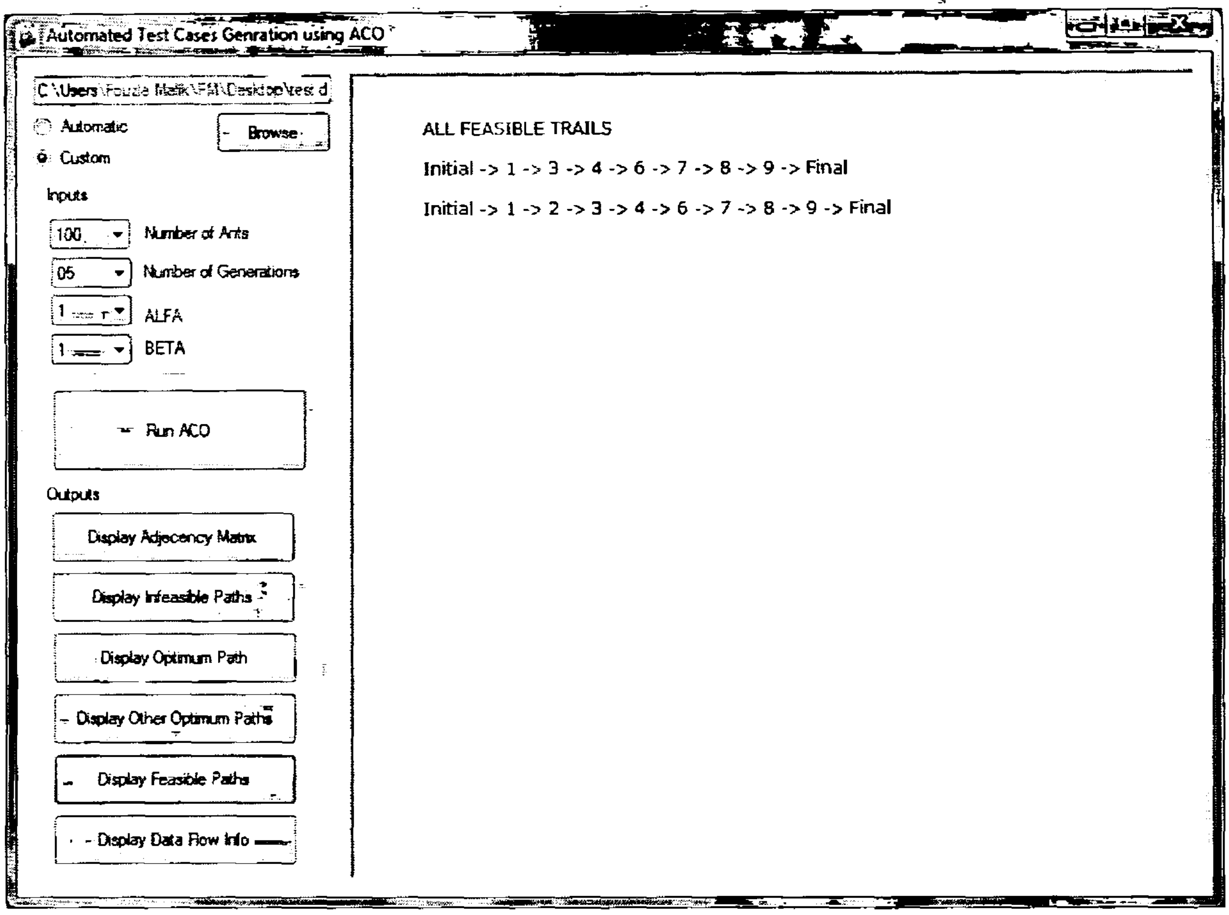
D.16 Display Adjacency Matrix of Student Enrollment System



100 Number of Ants
05 Number of Generations
1 ALFA
1 BETA

```
ADJECENCY MATRIX
0 1 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0
0 1 0 0 1 0 0 0 0 0 0
0 0 0 1 0 1 1 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 1 0 0 0
0 0 0 0 0 0 1 0 1 0 0
0 1 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0
```

D.17 Display Feasible Test Cases within SES Model



C:\Users\Fouzia Malik\FM\Desktop\test d

Automatic
[✓ Custom]

Inputs

100 Number of Ants

05 Number of Generations

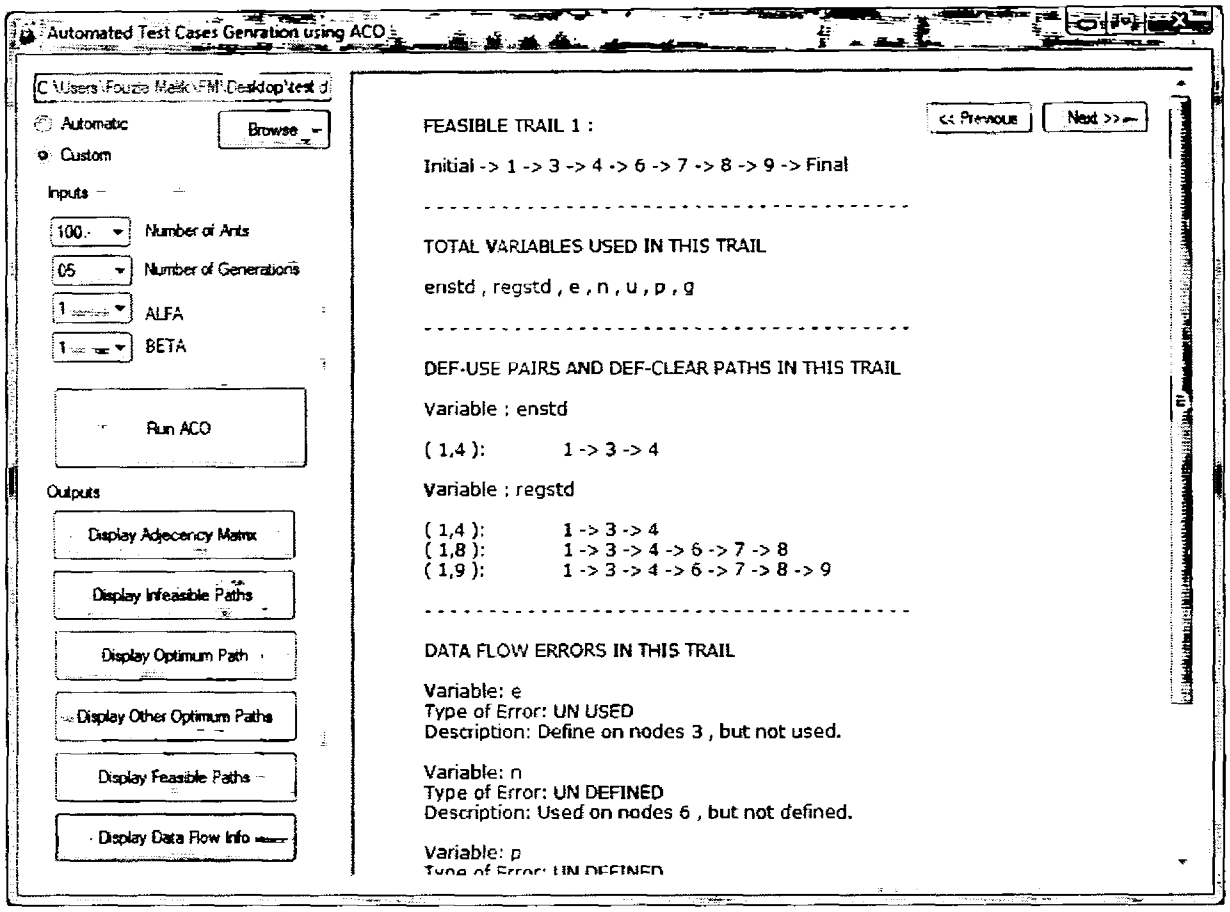
1 ALFA

1 BETA

Outputs

```
ALL FEASIBLE TRAILS

Initial -> 1 -> 3 -> 4 -> 6 -> 7 -> 8 -> 9 -> Final

Initial -> 1 -> 2 -> 3 -> 4 -> 6 -> 7 -> 8 -> 9 -> Final
```

D.18 Display All Def-Use Pairs & All Def-Use Paths within SES Model


Automated Test Cases Genration using ACO
C:\Users\Fouzia Malik\FM\Desktop\test d
Automatic
Custom
Browse
Inputs
Outputs
<< Previous
Next >>

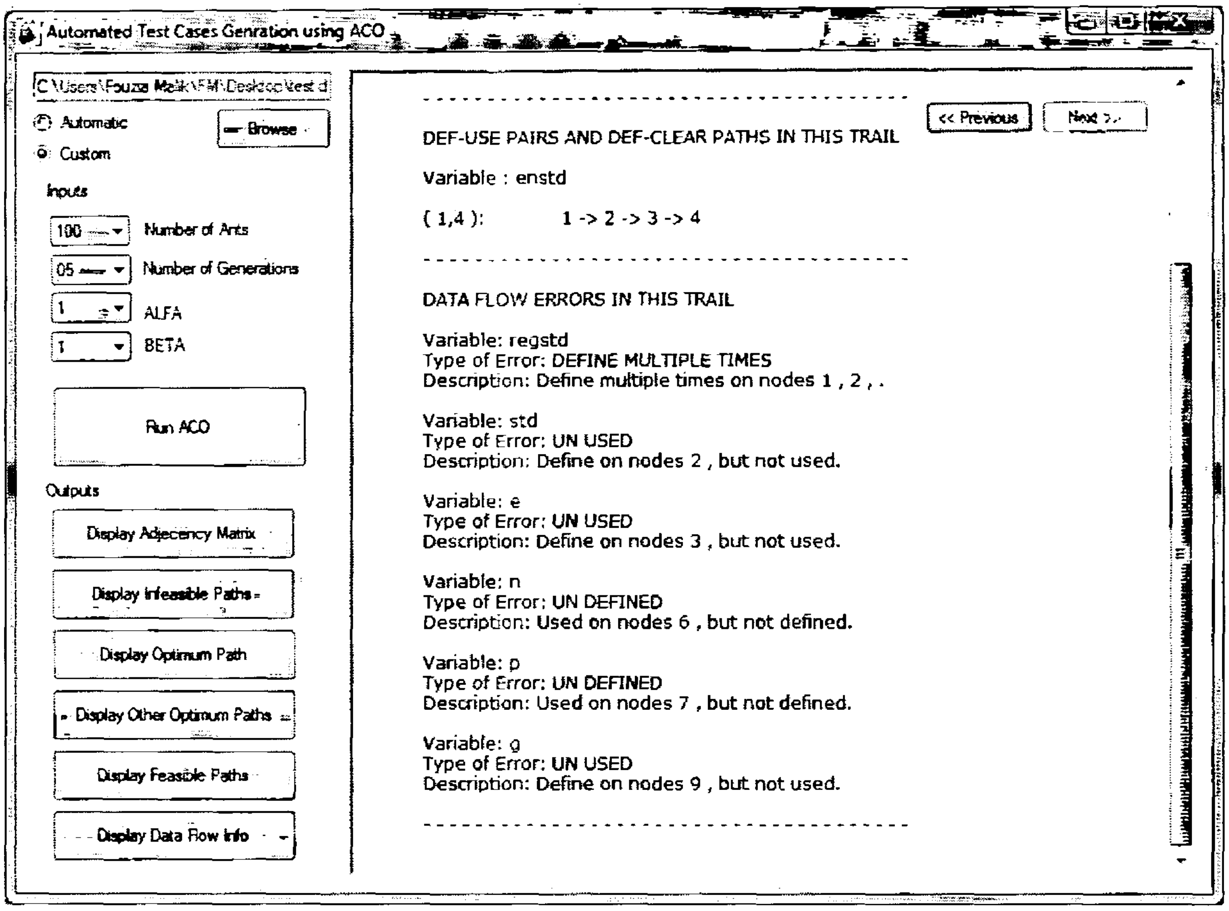
100 Number of Ants
05 Number of Generations
1 ALFA
1 BETA

Run ACO

Display Adjecency Matrix
Display Infeasible Paths
Display Optimum Path
Display Other Optimum Paths
Display Feasible Paths
Display Data Flow Info

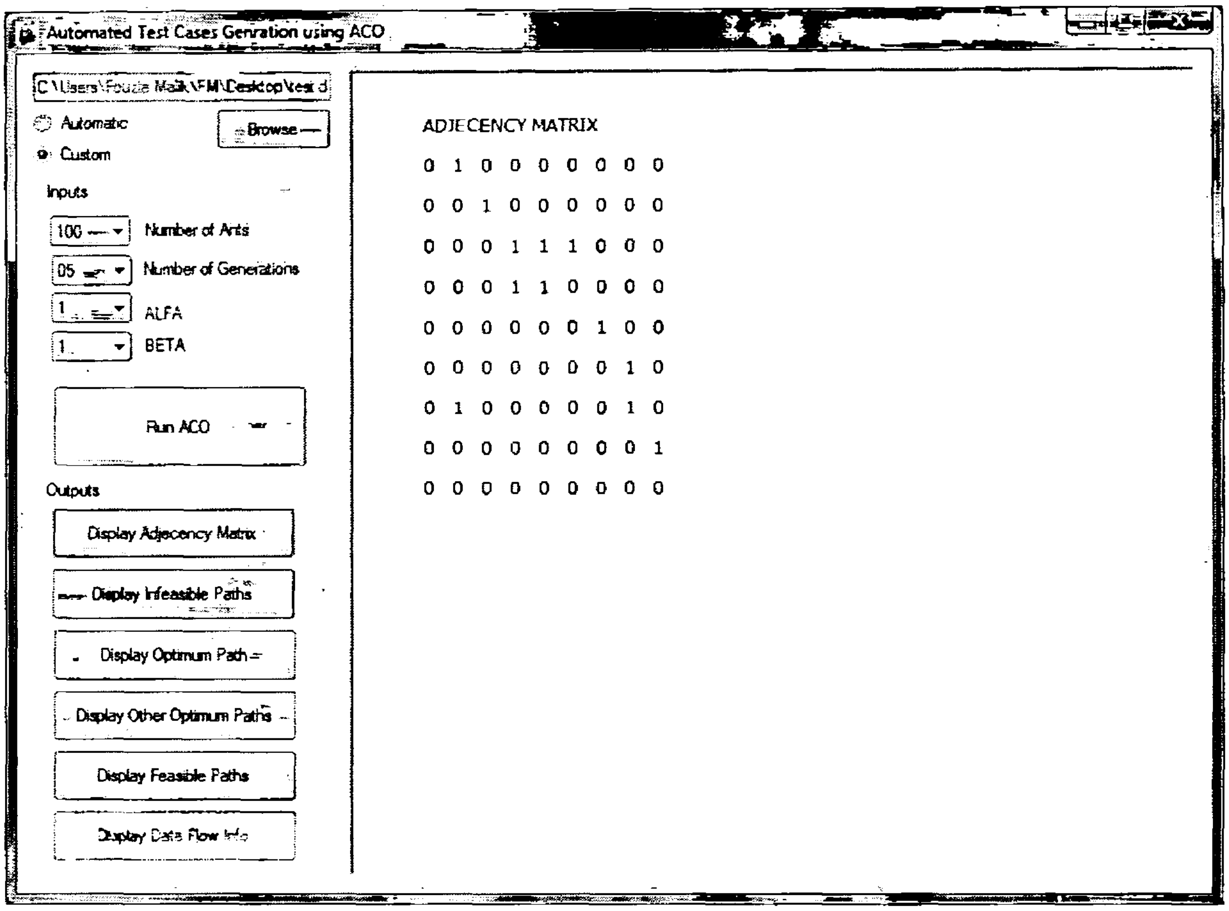
```
FEASIBLE TRAIL 1 :

Initial -> 1 -> 3 -> 4 -> 6 -> 7 -> 8 -> 9 -> Final

-----------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

enstd , regstd , e , n , u , p , g

-----------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : enstd

( 1,4 ):        1 -> 3 -> 4

Variable : regstd

( 1,4 ):        1 -> 3 -> 4
( 1,8 ):        1 -> 3 -> 4 -> 6 -> 7 -> 8
( 1,9 ):        1 -> 3 -> 4 -> 6 -> 7 -> 8 -> 9

-----------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: e
Type of Error: UN USED
Description: Define on nodes 3 , but not used.

Variable: n
Type of Error: UN DEFINED
Description: Used on nodes 6 , but not defined.

Variable: p
```

D.19 Display Data Flow Errors within SES Model



```
------------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : enstd

( 1,4 ):          1 -> 2 -> 3 -> 4

------------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: regstd
Type of Error: DEFINE MULTIPLE TIMES
Description: Define multiple times on nodes 1 , 2 , .

Variable: std
Type of Error: UN USED
Description: Define on nodes 2 , but not used.

Variable: e
Type of Error: UN USED
Description: Define on nodes 3 , but not used.

Variable: n
Type of Error: UN DEFINED
Description: Used on nodes 6 , but not defined.

Variable: p
Type of Error: UN DEFINED
Description: Used on nodes 7 , but not defined.

Variable: g
Type of Error: UN USED
Description: Define on nodes 9 , but not used.

------------------------------------
```

D.20 Display Adjacency Matrix of Purchase Order System



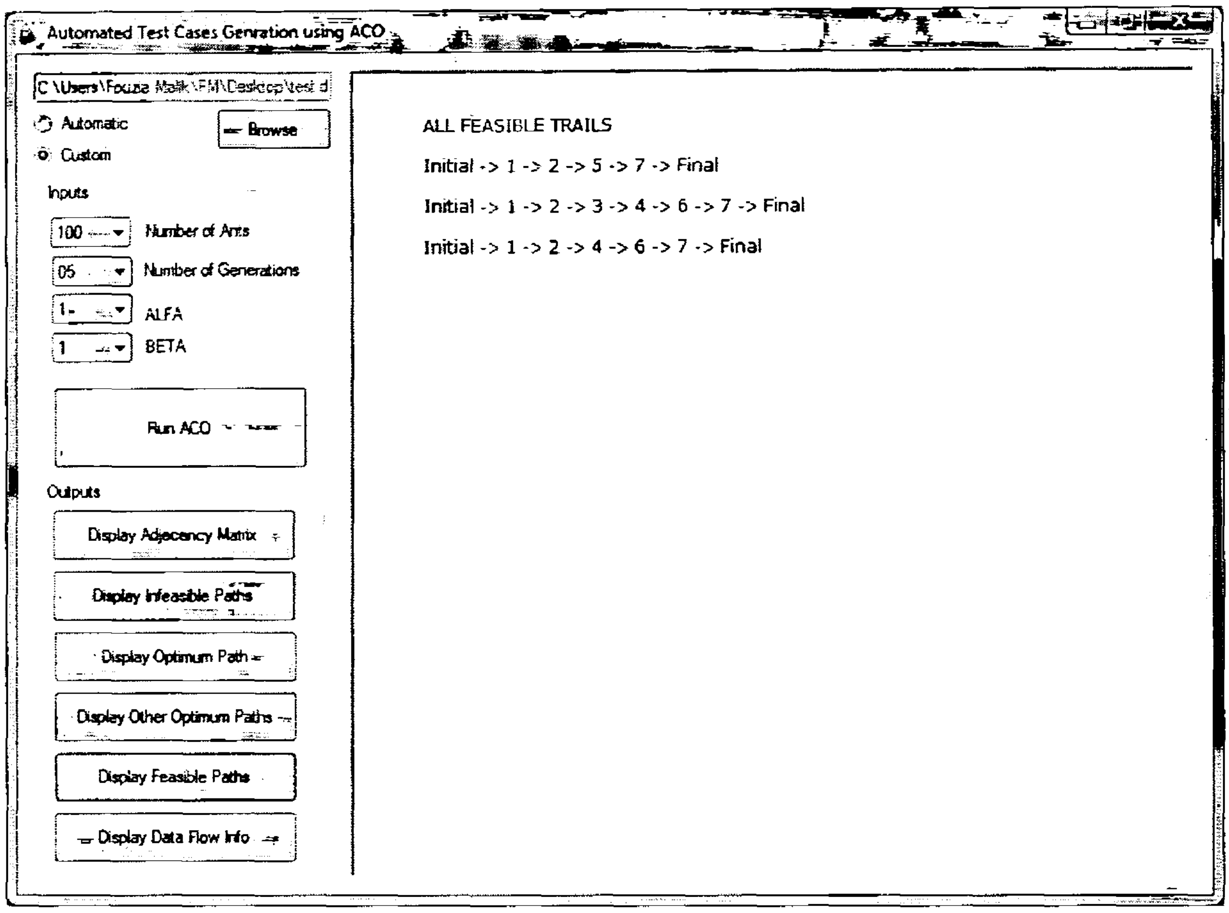
C:\Users\Fouzia Malik\FM\Desktop\test.d

Automatic
[✓ Custom]

100 Number of Ants
05 Number of Generations
1 ALFA
1 BETA

```
ADJECENCY MATRIX

0 1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0
0 0 0 1 1 0 0 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0
```

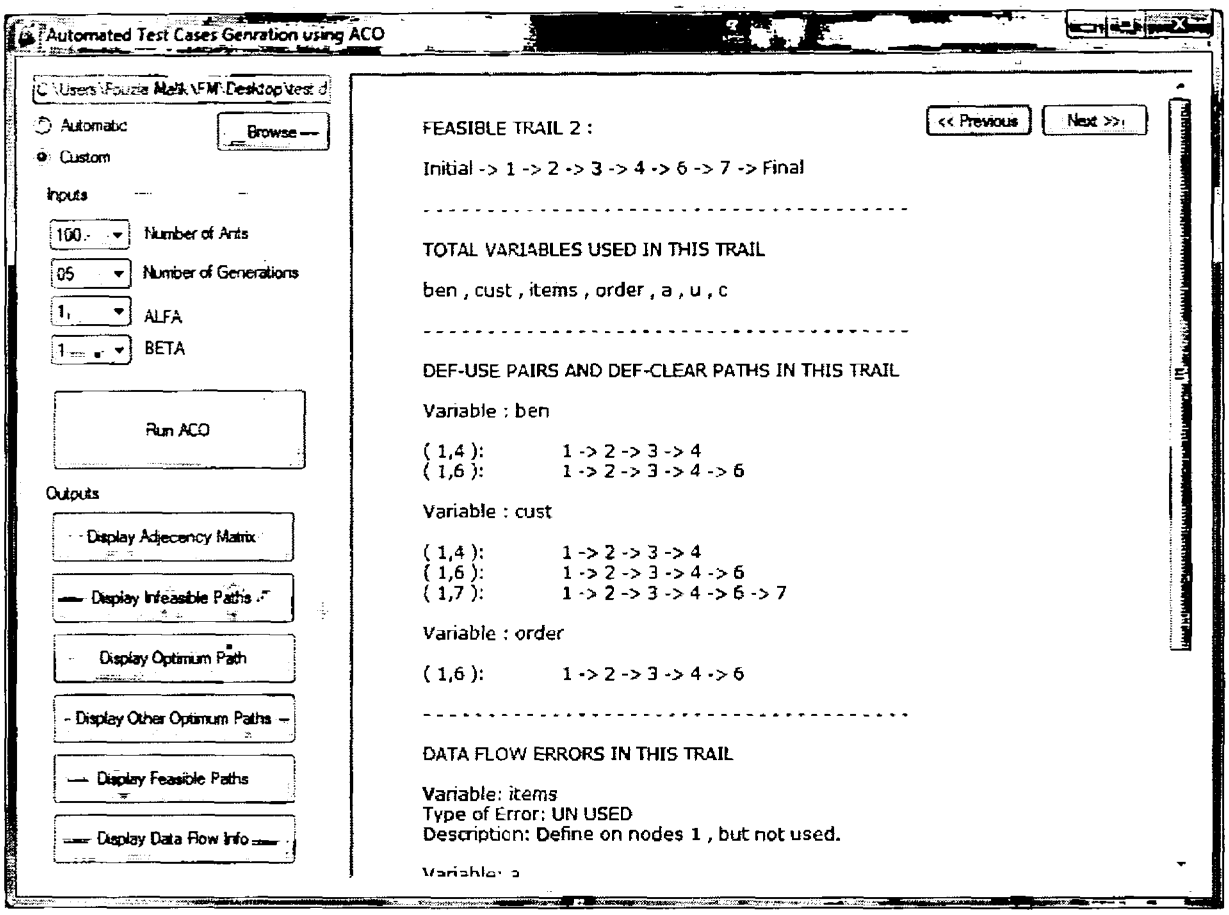
D.21 Display Feasible Test Cases within PO System



100 Number of Ants
05 Number of Generations
1 ALFA
1 BETA
Run ACO

```
ALL FEASIBLE TRAILS
Initial -> 1 -> 2 -> 5 -> 7 -> Final
Initial -> 1 -> 2 -> 3 -> 4 -> 6 -> 7 -> Final
Initial -> 1 -> 2 -> 4 -> 6 -> 7 -> Final
```

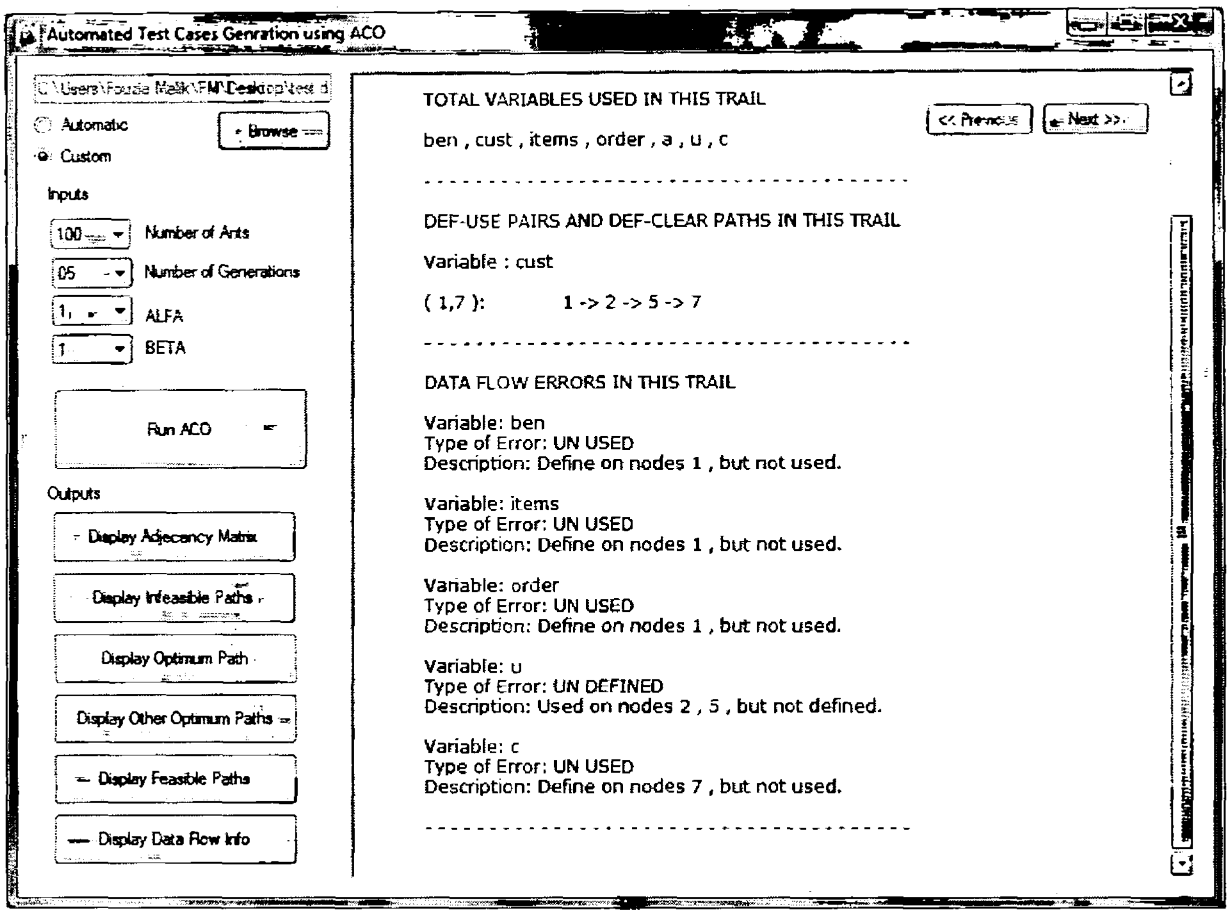
D.22 Display All Def-Use Paths & All Def-Use Pairs within PO System



100	Number of Ants
05	Number of Generations
1	ALFA
1	BETA

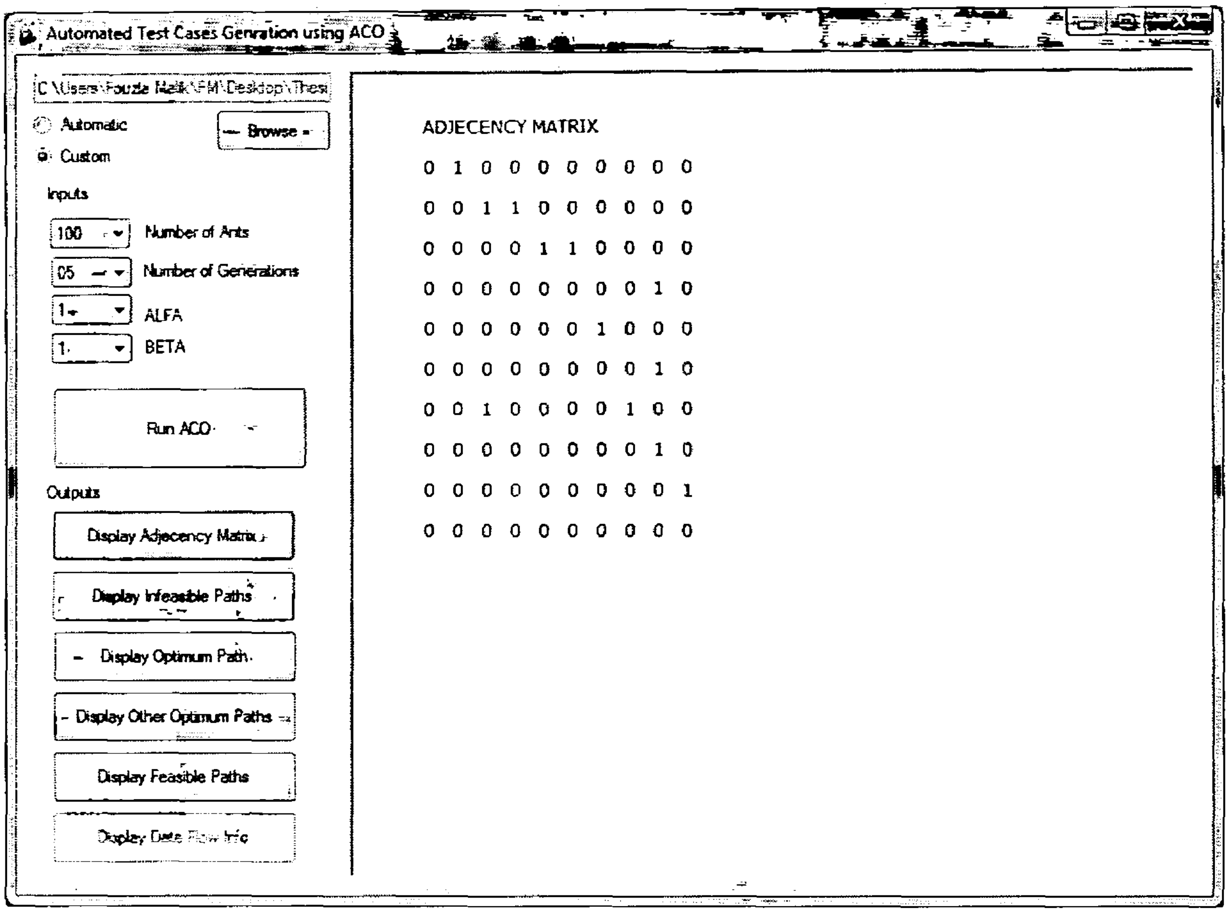
```
FEASIBLE TRAIL 2 :

Initial -> 1 -> 2 -> 3 -> 4 -> 6 -> 7 -> Final

-----------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

ben , cust , items , order , a , u , c

-----------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : ben

( 1,4 ):        1 -> 2 -> 3 -> 4
( 1,6 ):        1 -> 2 -> 3 -> 4 -> 6

Variable : cust

( 1,4 ):        1 -> 2 -> 3 -> 4
( 1,6 ):        1 -> 2 -> 3 -> 4 -> 6
( 1,7 ):        1 -> 2 -> 3 -> 4 -> 6 -> 7

Variable : order

( 1,6 ):        1 -> 2 -> 3 -> 4 -> 6

-----------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: items
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: a
```

D.23 Display Data Flow Errors within PO System



```
TOTAL VARIABLES USED IN THIS TRAIL

ben , cust , items , order , a , u , c

-------------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : cust

( 1,7 ):          1 -> 2 -> 5 -> 7

-------------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: ben
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: items
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: order
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: u
Type of Error: UN DEFINED
Description: Used on nodes 2 , 5 , but not defined.

Variable: c
Type of Error: UN USED
Description: Define on nodes 7 , but not used.

-------------------------------------
```

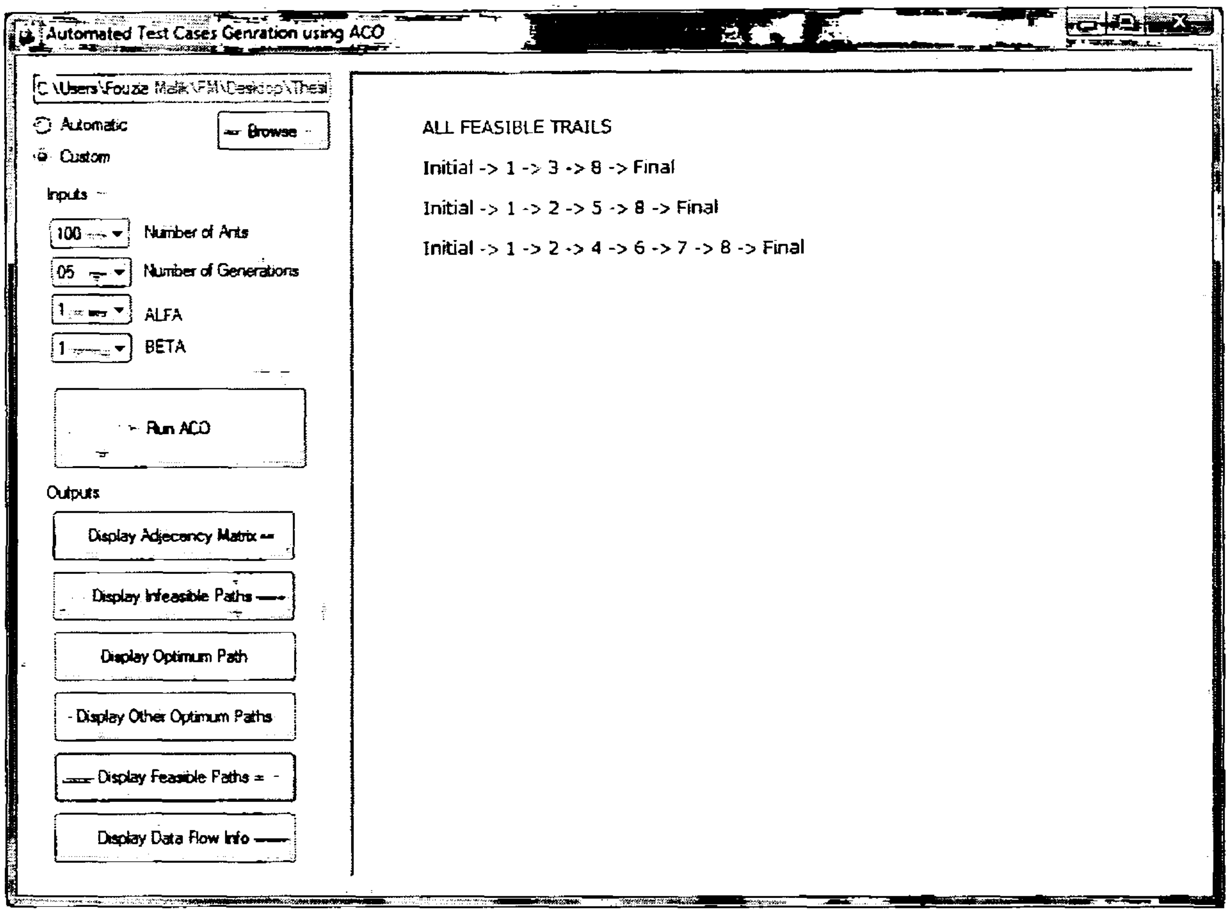
D.24 Display Adjacency Matrix of Hotel Management System



Number of Ants: 100
Number of Generations: 05
ALFA: 1
BETA: 1

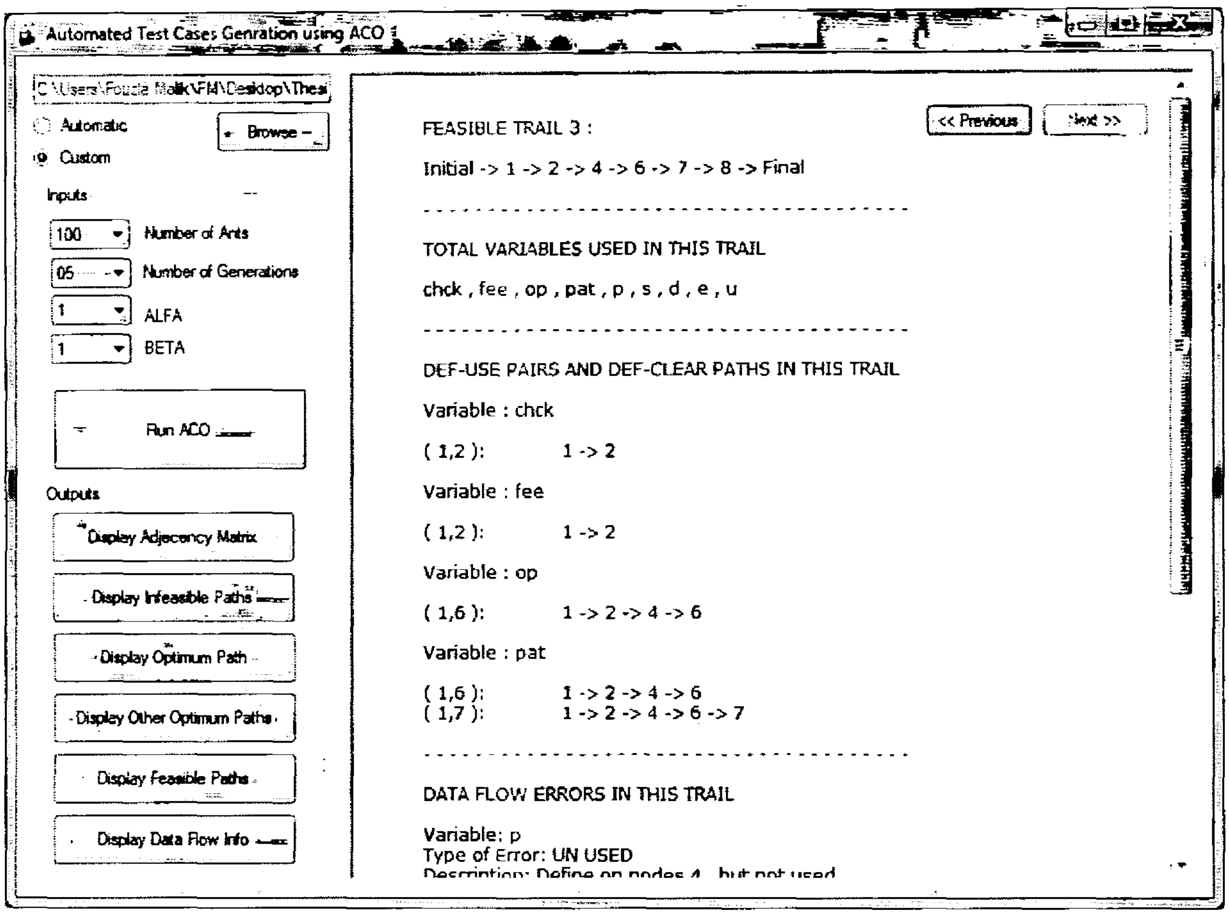
```
ADJECENCY MATRIX
0 1 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0
0 0 0 0 1 1 0 0 0 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 1 0
0 0 1 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0
```

D.25 Display Feasible Test Cases within HMS



```
ALL FEASIBLE TRAILS
Initial -> 1 -> 3 -> 8 -> Final
Initial -> 1 -> 2 -> 5 -> 8 -> Final
Initial -> 1 -> 2 -> 4 -> 6 -> 7 -> 8 -> Final
```

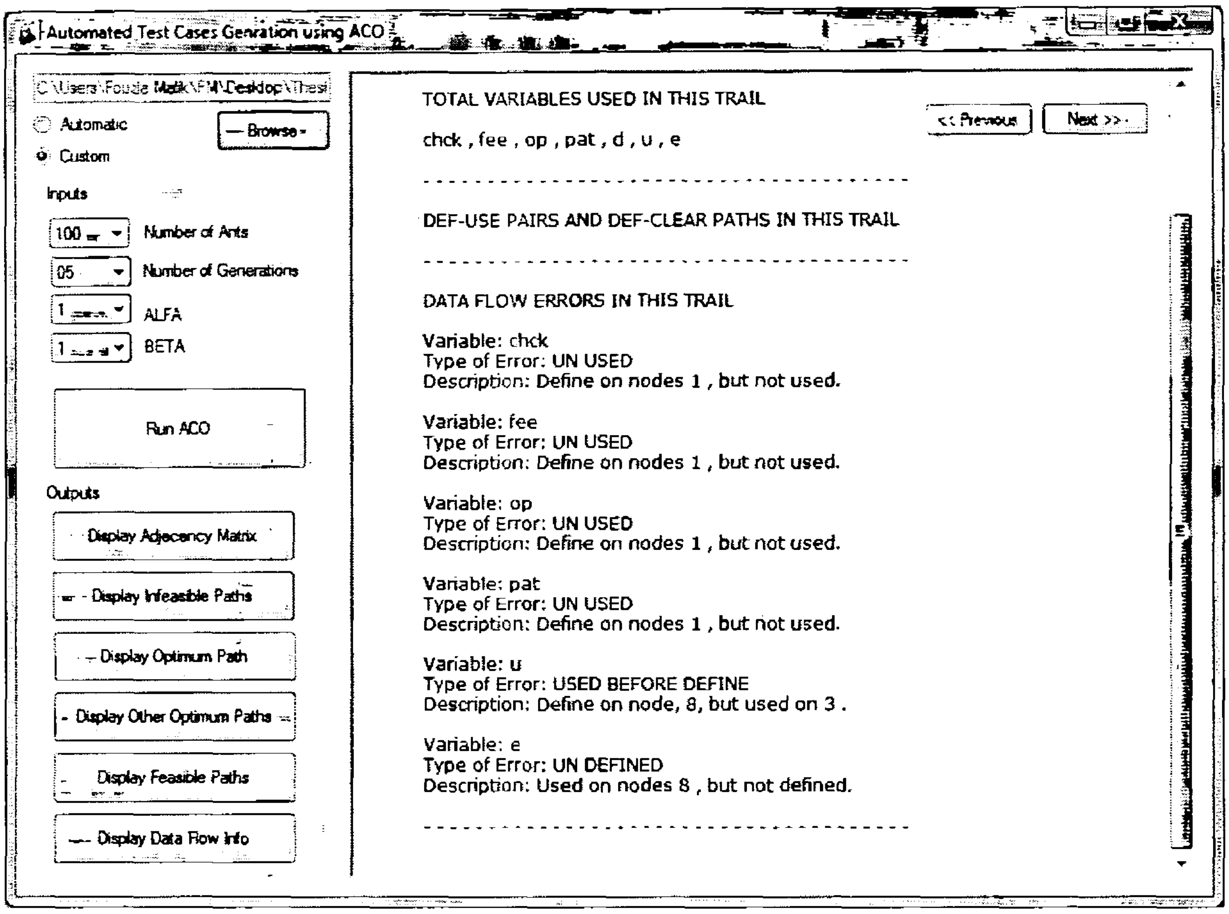
D.26 Display All Def-Use Pairs & All Def-Use Paths within HMS



Number of Ants: 100
Number of Generations: 05
ALFA: 1
BETA: 1

```
FEASIBLE TRAIL 3 :

Initial -> 1 -> 2 -> 4 -> 6 -> 7 -> 8 -> Final

-------------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

chck , fee , op , pat , p , s , d , e , u

-------------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : chck

( 1,2 ):        1 -> 2

Variable : fee

( 1,2 ):        1 -> 2

Variable : op

( 1,6 ):        1 -> 2 -> 4 -> 6

Variable : pat

( 1,6 ):        1 -> 2 -> 4 -> 6
( 1,7 ):        1 -> 2 -> 4 -> 6 -> 7

-------------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: p
Type of Error: UN USED
Description: Define on nodes 4   but not used
```

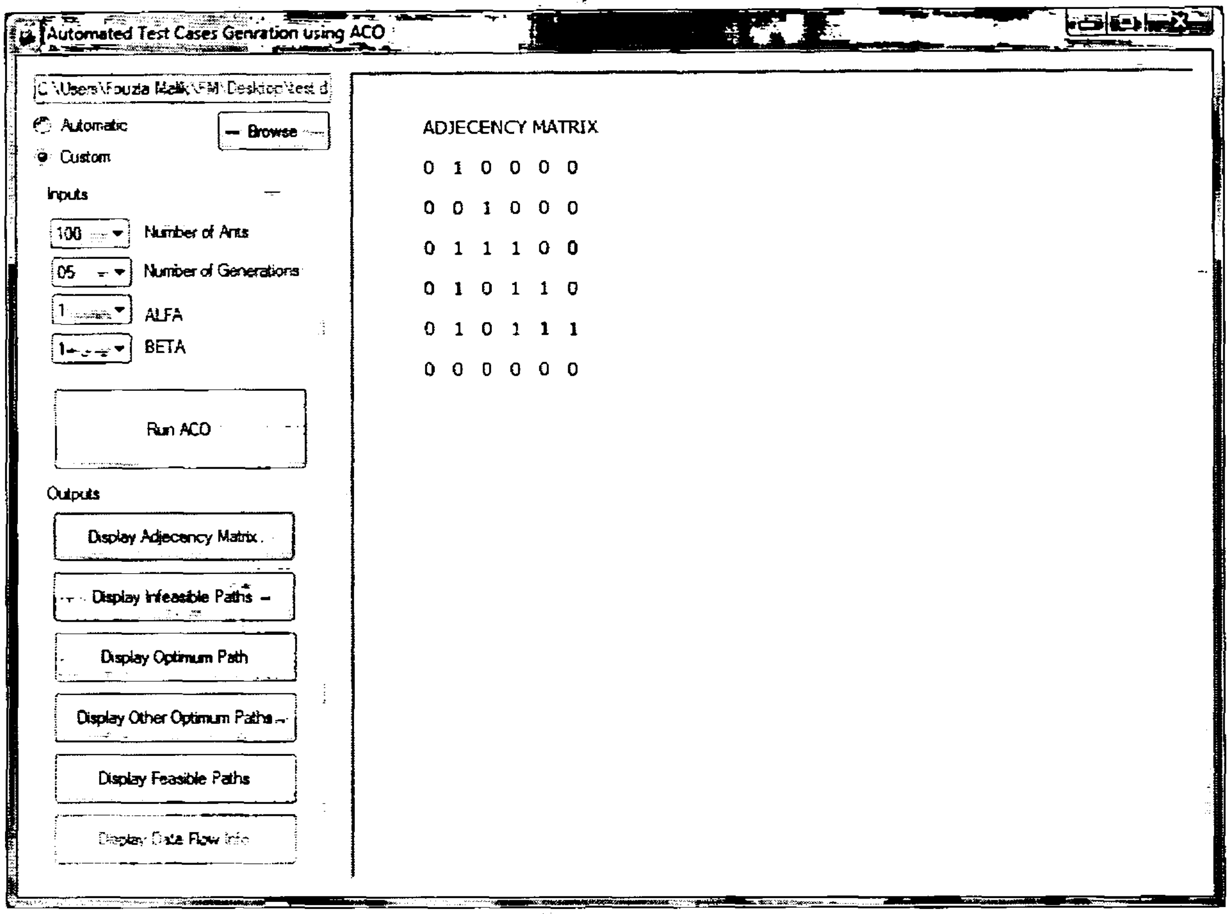
D.27 Display Data Flow Errors within HMS



Number of Ants: 100
Number of Generations: 05
ALFA: 1
BETA: 1

```
TOTAL VARIABLES USED IN THIS TRAIL

chck , fee , op , pat , d , u , e

-----------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

-----------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: chck
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: fee
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: op
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: pat
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: u
Type of Error: USED BEFORE DEFINE
Description: Define on node, 8, but used on 3 .

Variable: e
Type of Error: UN DEFINED
Description: Used on nodes 8 , but not defined.

-----------------------------------
```

D.28 Display Adjacency Matrix of Cruise Control Model



Custom [✓]

100 Number of Ants

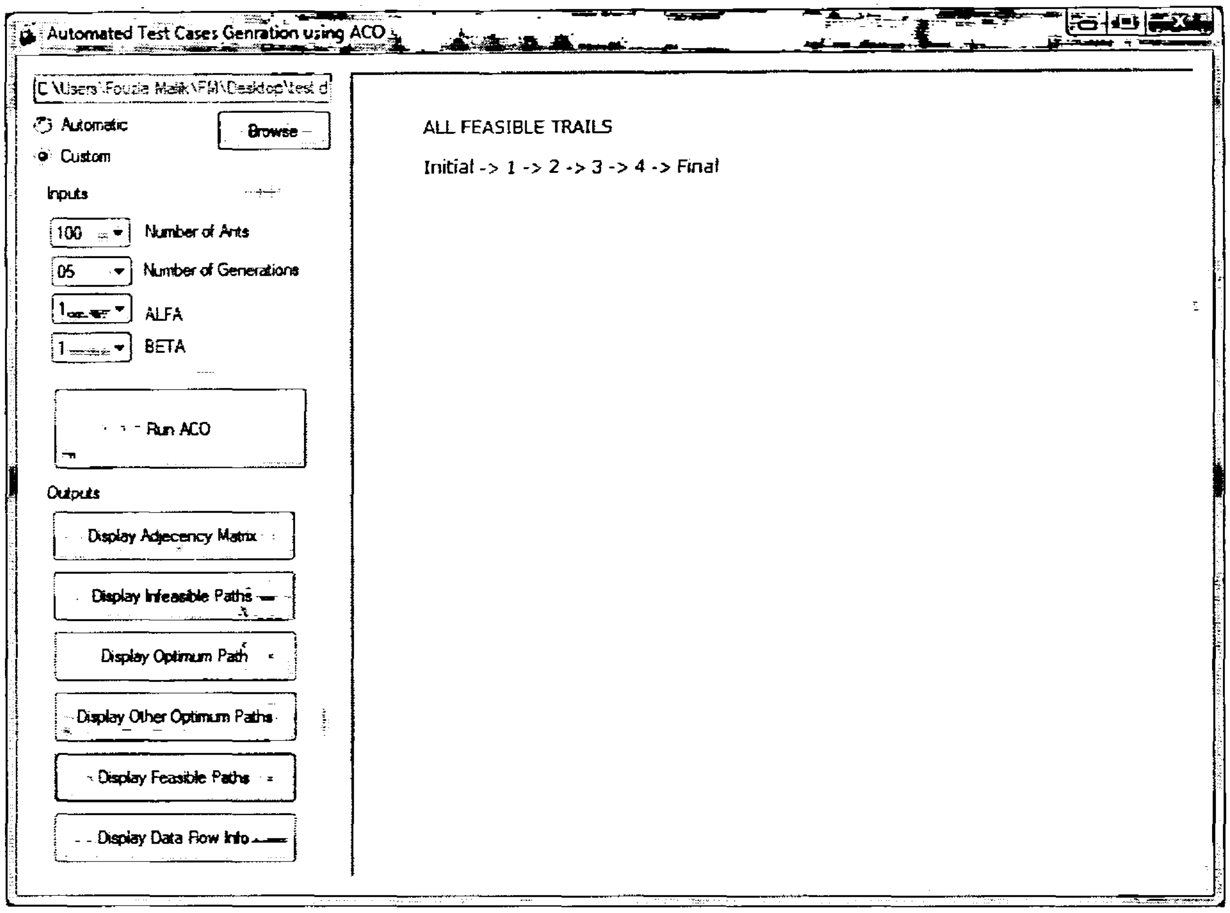
05 Number of Generations

1 ALFA

1 BETA

```
ADJECENCY MATRIX
0  1  0  0  0  0
0  0  1  0  0  0
0  1  1  1  0  0
0  1  0  1  1  0
0  1  0  1  1  1
0  0  0  0  0  0
```

D.29 Display Feasible Paths within CC Model



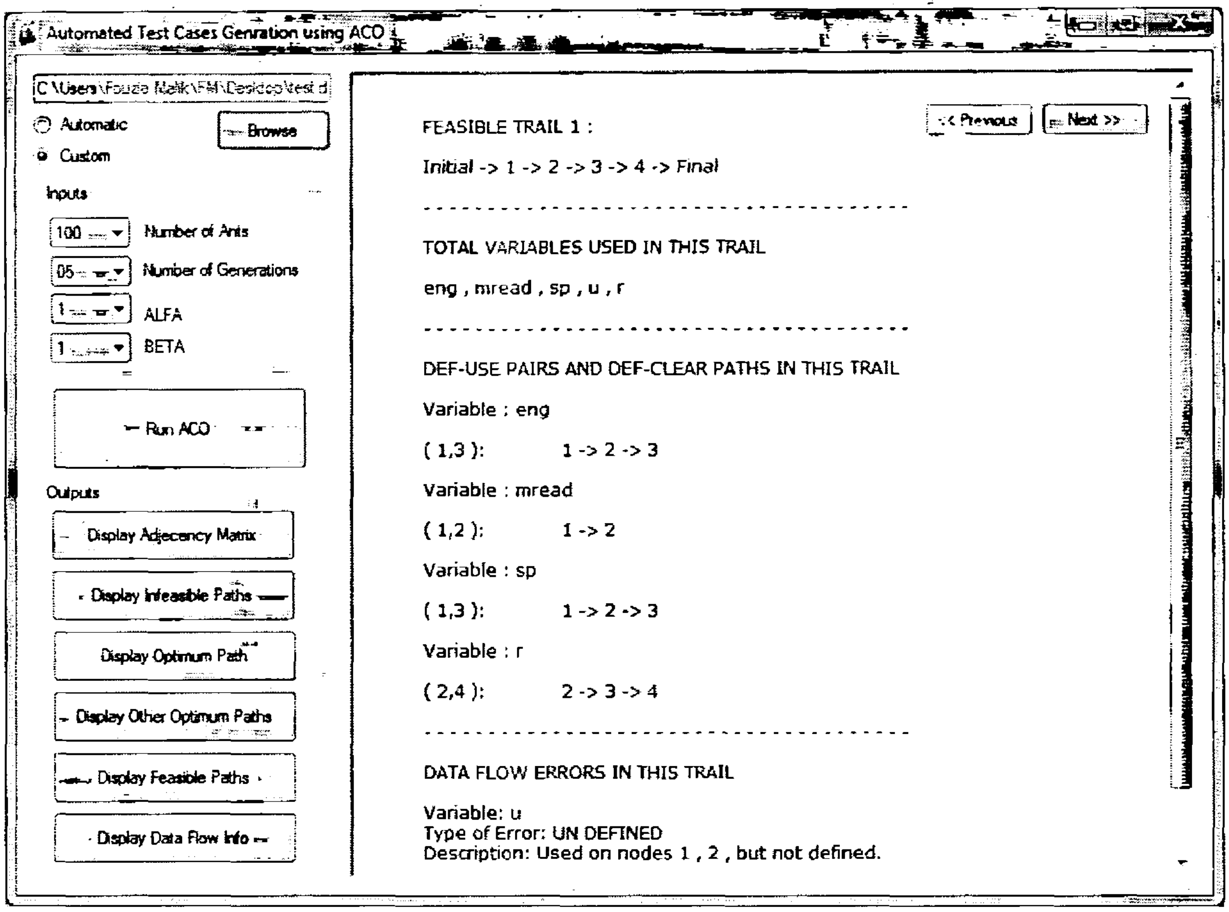
Custom selected

100 Number of Ants
05 Number of Generations
1 ALFA
1 BETA

ALL FEASIBLE TRAILS

Initial -> 1 -> 2 -> 3 -> 4 -> Final

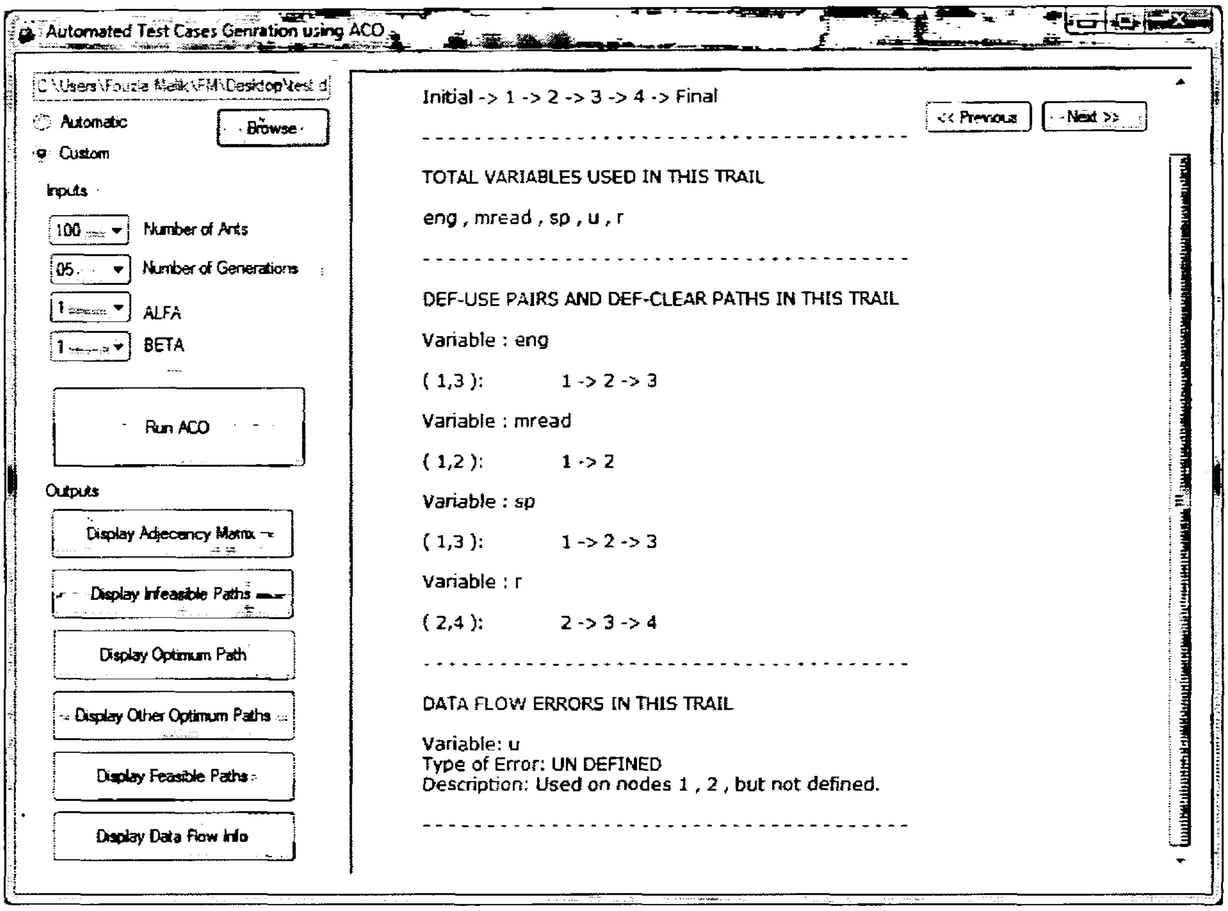
D. 30 Display All Def-Use Pairs & All Def-Use Paths within CC Model



100 Number of Ants
05 Number of Generations
1 ALFA
1 BETA

```
FEASIBLE TRAIL 1 :

Initial -> 1 -> 2 -> 3 -> 4 -> Final

-------------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

eng , mread , sp , u , r

-------------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : eng

( 1,3 ):        1 -> 2 -> 3

Variable : mread

( 1,2 ):        1 -> 2

Variable : sp

( 1,3 ):        1 -> 2 -> 3

Variable : r

( 2,4 ):        2 -> 3 -> 4

-------------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: u
Type of Error: UN DEFINED
Description: Used on nodes 1 , 2 , but not defined.
```

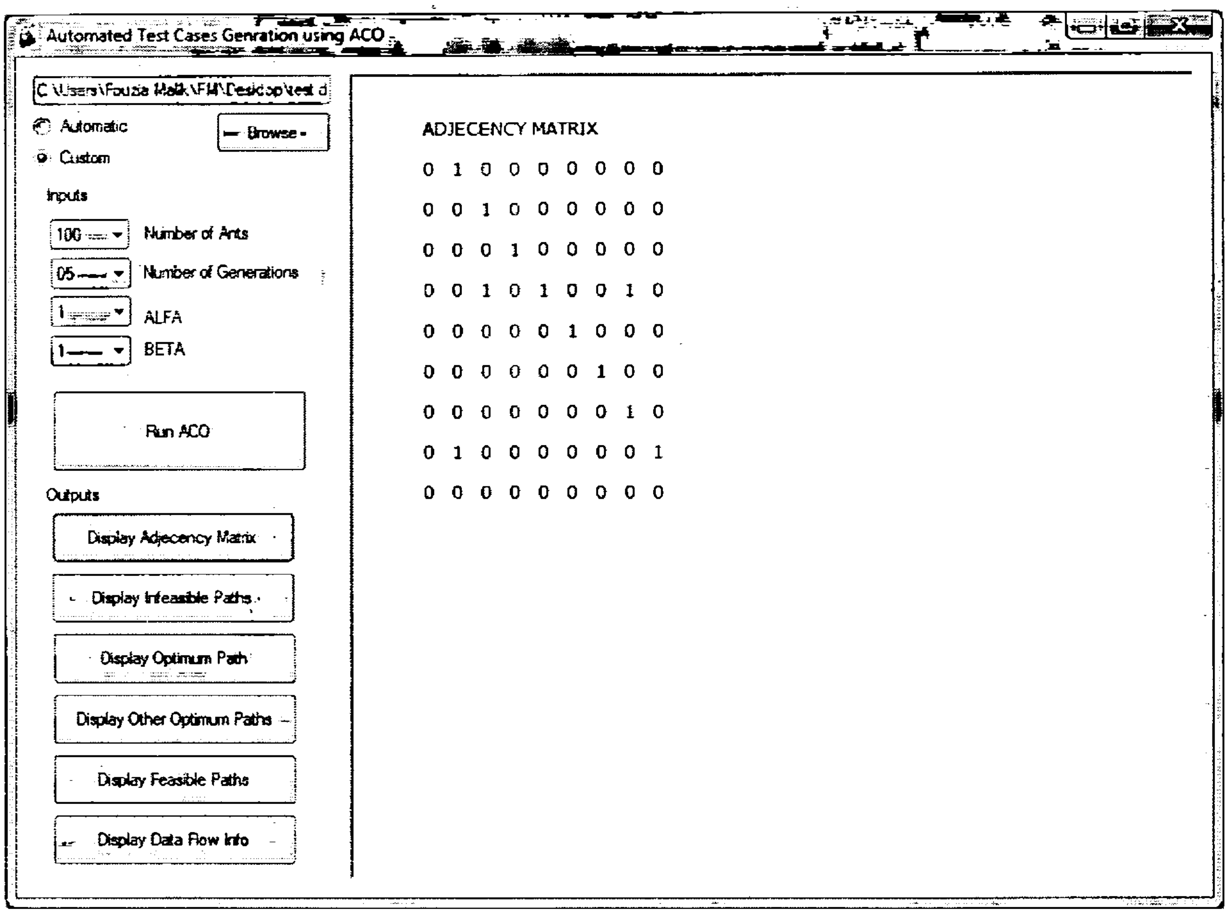
D.31 Display Data Flow Errors within CC Model



Custom: 100 Number of Ants; 05 Number of Generations; 1 ALFA; 1 BETA

```
Initial -> 1 -> 2 -> 3 -> 4 -> Final
------------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

eng , mread , sp , u , r
------------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : eng

( 1,3 ):        1 -> 2 -> 3

Variable : mread

( 1,2 ):        1 -> 2

Variable : sp

( 1,3 ):        1 -> 2 -> 3

Variable : r

( 2,4 ):        2 -> 3 -> 4
------------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: u
Type of Error: UN DEFINED
Description: Used on nodes 1 , 2 , but not defined.
------------------------------------
```

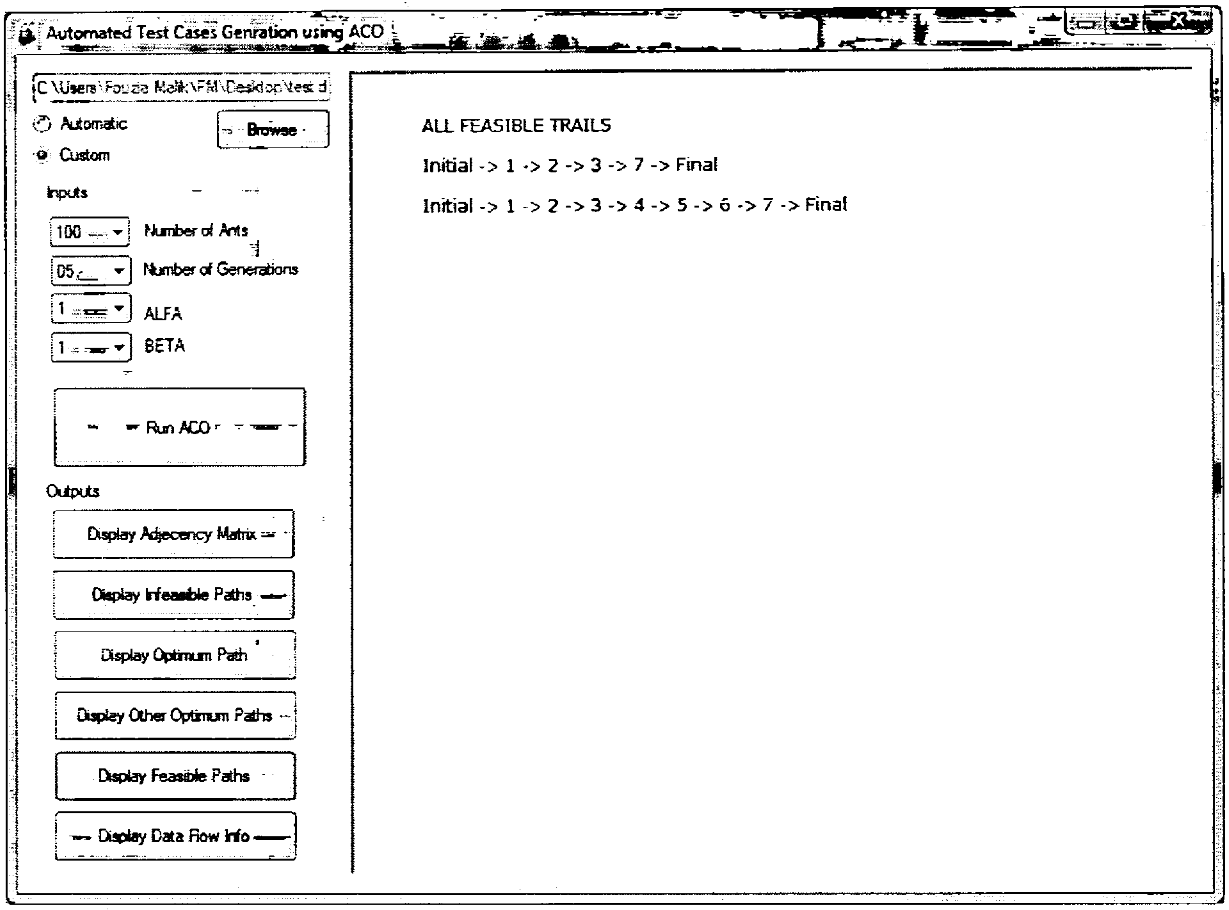
D.32 Display Adjacency Matrix of ATM Model



Inputs: 100 Number of Ants; 05 Number of Generations; 1 ALFA; 1 BETA

```
ADJECENCY MATRIX
0 1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0
0 0 1 0 1 0 0 1 0
0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0
```

D.33 Display Feasible Test Cases within ATM Model



Custom [✓]

100 Number of Ants

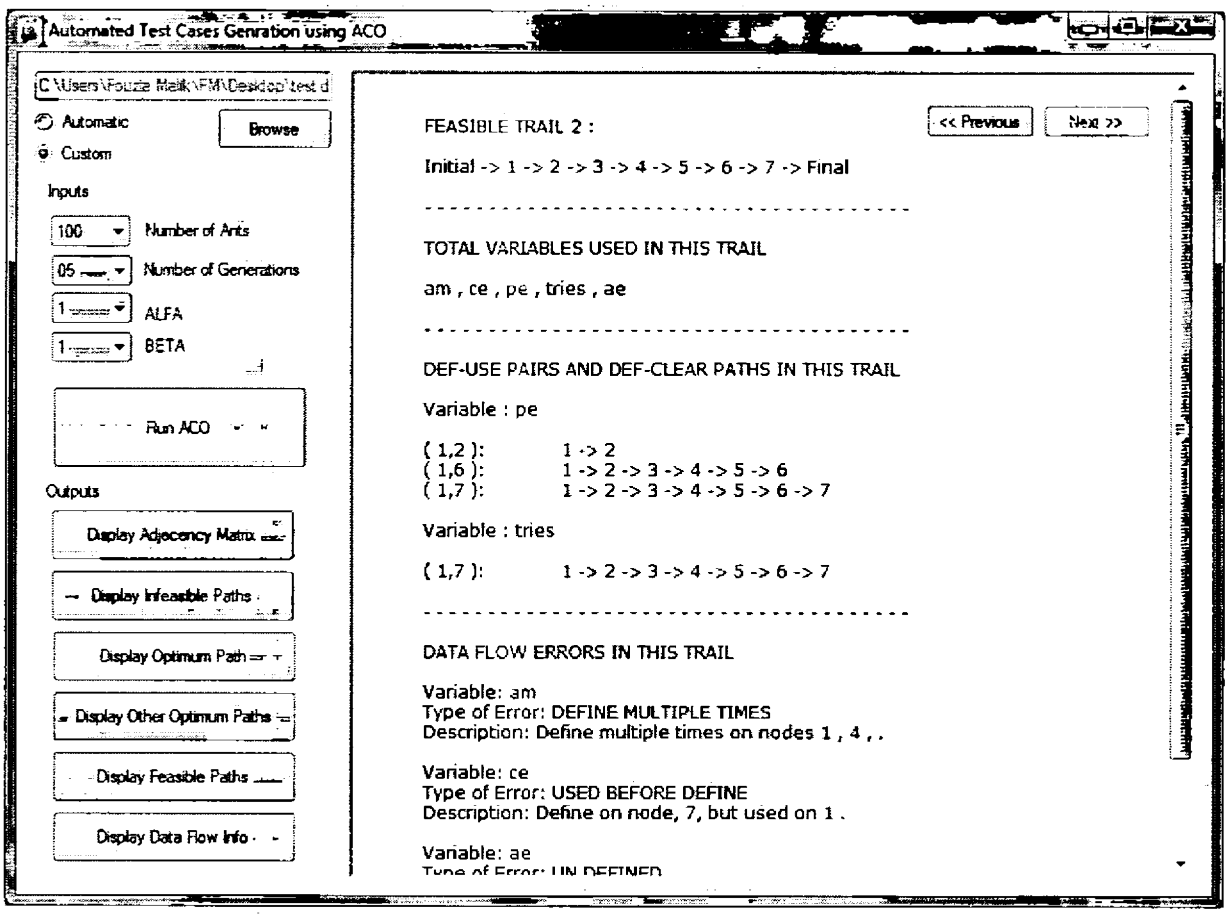
05 Number of Generations

1 ALFA

1 BETA

```
ALL FEASIBLE TRAILS

Initial -> 1 -> 2 -> 3 -> 7 -> Final

Initial -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> Final
```

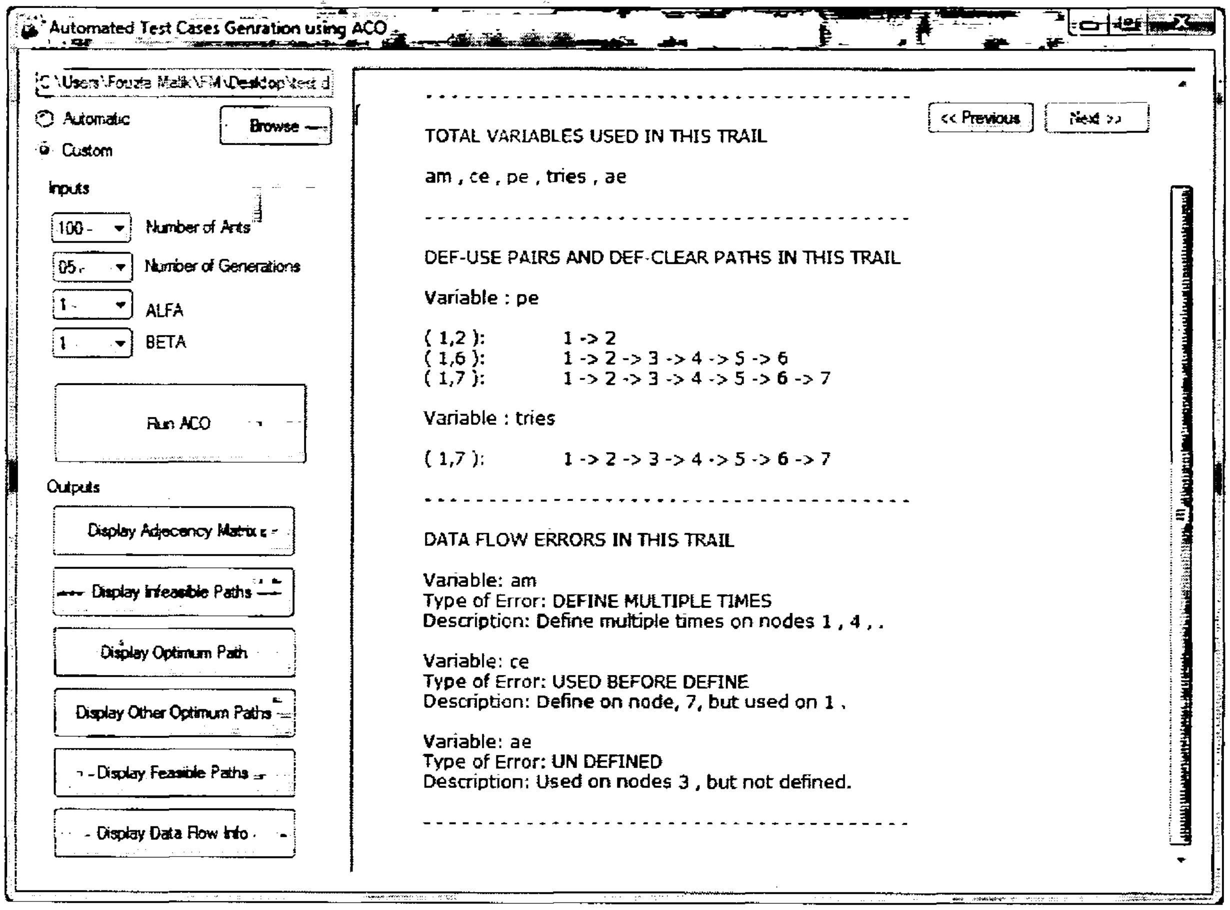
D.34 Display All Def-Use Pairs & All Def-Use Paths within ATM Model



100 Number of Ants
05 Number of Generations
1 ALFA
1 BETA

```
FEASIBLE TRAIL 2 :

Initial -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> Final

------------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

am , ce , pe , tries , ae

------------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : pe

( 1,2 ):        1 -> 2
( 1,6 ):        1 -> 2 -> 3 -> 4 -> 5 -> 6
( 1,7 ):        1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7

Variable : tries

( 1,7 ):        1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7

------------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: am
Type of Error: DEFINE MULTIPLE TIMES
Description: Define multiple times on nodes 1 , 4 , .

Variable: ce
Type of Error: USED BEFORE DEFINE
Description: Define on node, 7, but used on 1 .

Variable: ae
```

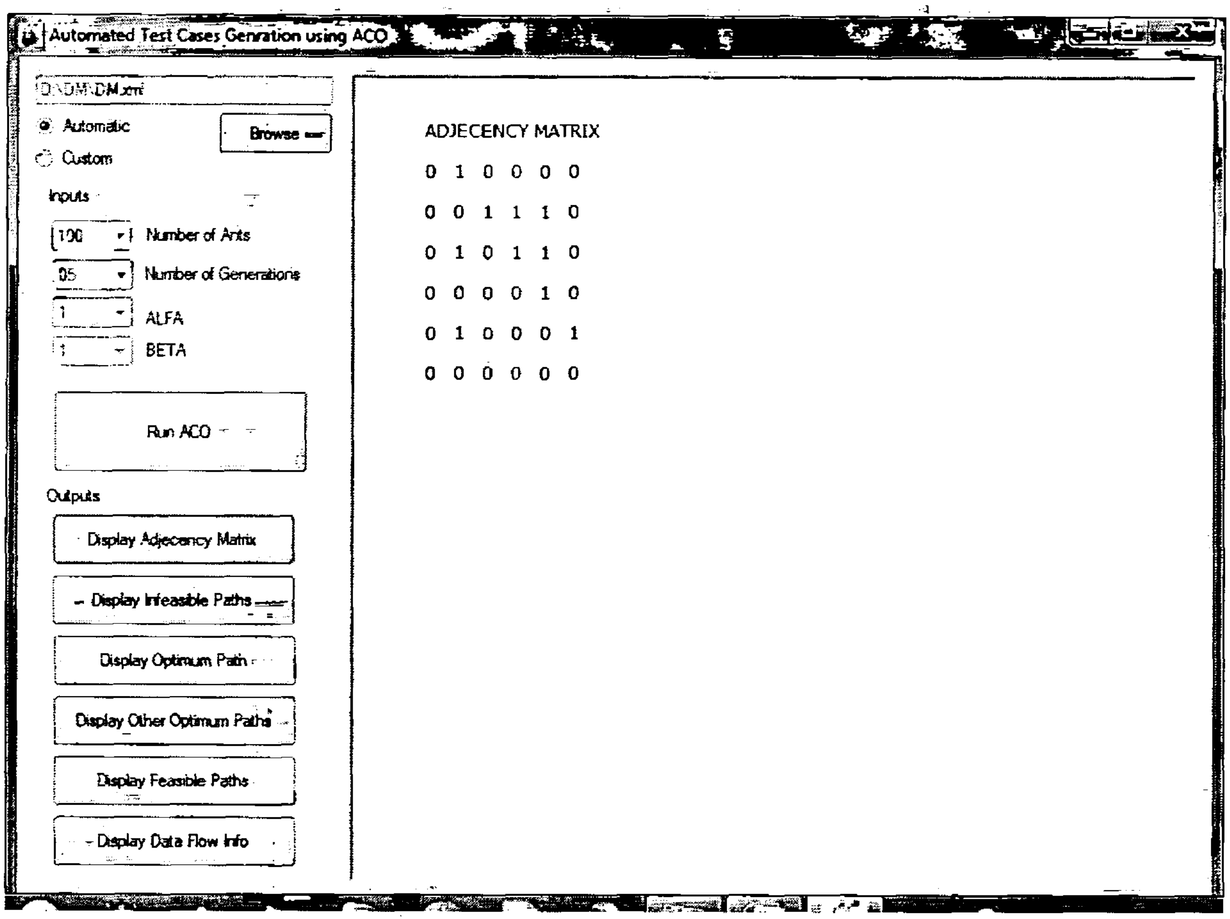
D.35 Display Data Flow Errors within ATM Model



Inputs: Number of Ants 100; Number of Generations 05; ALFA 1; BETA 1

```
-----------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

am , ce , pe , tries , ae

-----------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

Variable : pe

( 1,2 ):        1 -> 2
( 1,6 ):        1 -> 2 -> 3 -> 4 -> 5 -> 6
( 1,7 ):        1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7

Variable : tries

( 1,7 ):        1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7

-----------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: am
Type of Error: DEFINE MULTIPLE TIMES
Description: Define multiple times on nodes 1 , 4 , .

Variable: ce
Type of Error: USED BEFORE DEFINE
Description: Define on node, 7, but used on 1 ,

Variable: ae
Type of Error: UN DEFINED
Description: Used on nodes 3 , but not defined.

-----------------------------------
```

D.36 Display Adjacency Matrix of Display Manager State Machine Model



D:\DM\DM.xmi

Number of Ants: 100

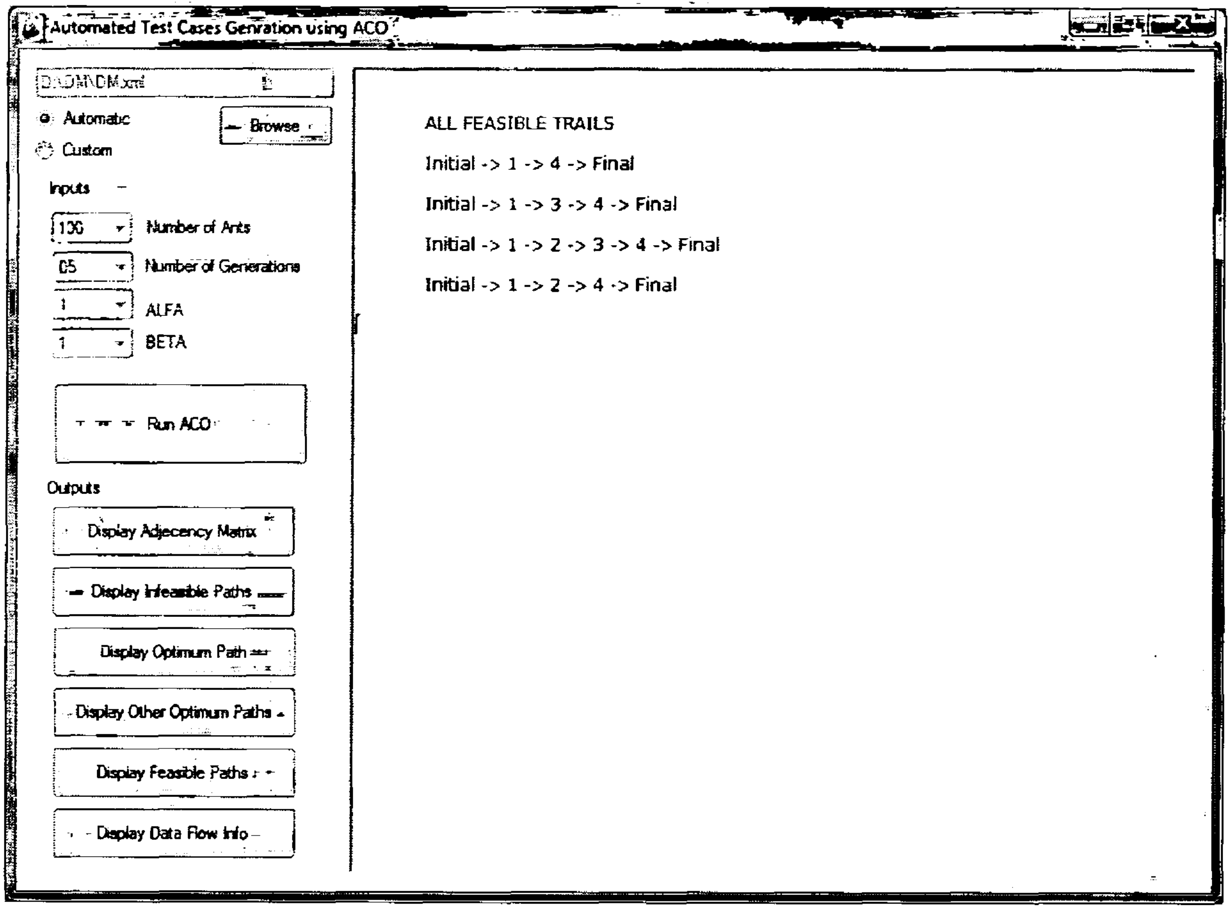
Number of Generations: 05

ALFA: 1

BETA: 1

```
ADJECENCY MATRIX

0  1  0  0  0  0

0  0  1  1  1  0

0  1  0  1  1  0

0  0  0  0  1  0

0  1  0  0  0  1

0  0  0  0  0  0
```

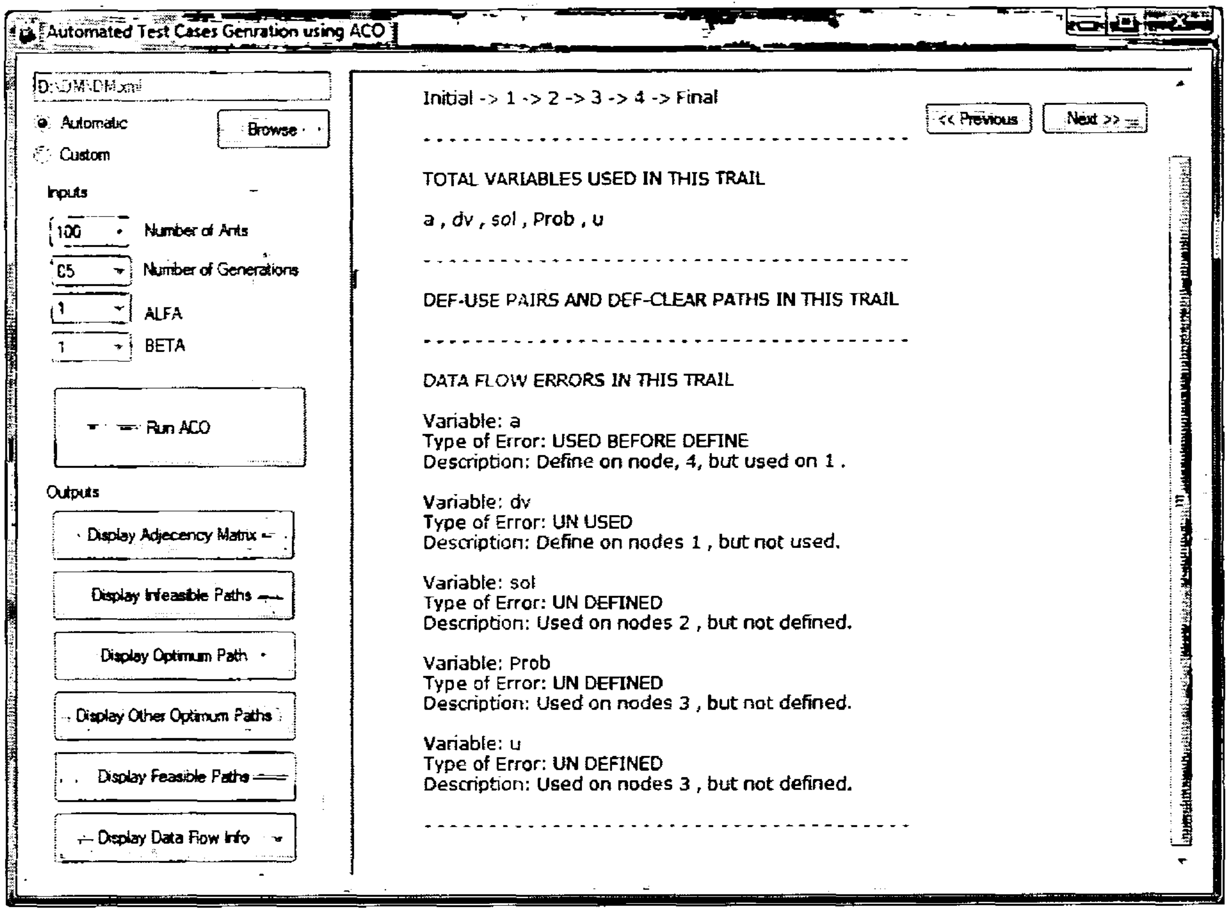
D.37 Feasible Test Cases of Display Manager State Machine Model



Number of Ants: 100
Number of Generations: 05
ALFA: 1
BETA: 1

```
ALL FEASIBLE TRAILS
Initial -> 1 -> 4 -> Final
Initial -> 1 -> 3 -> 4 -> Final
Initial -> 1 -> 2 -> 3 -> 4 -> Final
Initial -> 1 -> 2 -> 4 -> Final
```

D.38 Data Flow Errors in Display Manager State Machine Model



```
Initial -> 1 -> 2 -> 3 -> 4 -> Final
------------------------------------

TOTAL VARIABLES USED IN THIS TRAIL

a , dv , sol , Prob , u

------------------------------------

DEF-USE PAIRS AND DEF-CLEAR PATHS IN THIS TRAIL

------------------------------------

DATA FLOW ERRORS IN THIS TRAIL

Variable: a
Type of Error: USED BEFORE DEFINE
Description: Define on node, 4, but used on 1 .

Variable: dv
Type of Error: UN USED
Description: Define on nodes 1 , but not used.

Variable: sol
Type of Error: UN DEFINED
Description: Used on nodes 2 , but not defined.

Variable: Prob
Type of Error: UN DEFINED
Description: Used on nodes 3 , but not defined.

Variable: u
Type of Error: UN DEFINED
Description: Used on nodes 3 , but not defined.

------------------------------------
```