LEBANESE AMERICAN UNIVERSITY

METAHEURISTIC ALGORITHM FOR TESTING WEB 2.0 APPLICATIONS

By

HRATCH MICHEL ZEITUNLIAN

A thesis
Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

School of Arts and Sciences
January 2012

Lebanese American University
School of Arts and Sciences

Thesis Proposal Form

Name of Student: ______Hratch Zeitunlian____________________ I.D.#: ____200402622____

Division: ________________Computer Science & Mathematics____________________________

On (dd/mm/yy) ______18/ 03/11_______________, has presented a Thesis proposal entitled:

_________________________Testing Web 2.0 Applications_____________________

____

in the presence of the Committee members and Thesis Advisor:

___Dr. Nashat Mansour ________ [redacted] 18/3/11 ____________
(Na[redacted]e of the Thesis Advisor)

___Dr. Sana Sharafeddine _______ 18/3/2011 __________
(Name, signature, and date of Committee Member)

__Dr. Abbas Tarhini_____________ [redacted] 18/3/2011 ____
(Name, signature, and date of Committee Member)

Comments/Remarks/Conditions to Proposal: Explore:

1. Adding significance weights
2. Ensuring coverage + diversity in "fitness" function
3. Min. # independent paths
4. Metaheuristic to consider weights

Date: 21/3/2011 Acknowledged by [redacted]________________
(Dean of G[redacted]l of ……………)

cc: Department Chair
Thesis Advisor
Student
/Dean of Graduate Studies/School Dean

Lebanese American University
School of Arts and Sciences

Thesis Defense Result Form

Name of Student: ___Hratch Zeitunlian__________________ I.D.#: 200402622

On (dd/mm/yy) __29/09/2011________________________, has defended a Thesis entitled:
_______Metaheuristic algorithm for testing web 2.0 applications________________________

In the presence of the following Committee members: [redacted]

Advisor	Nashat Mansour, 29/09/2011 (Name, signature, and date)
Committee Member	Sanaa Sharafeddine, 29/09/2011 (Name, signature, and date)
Committee Member	Abbas Tarhini, 29/09/2011 (Name, signature, and date)

The student has [✓ passed] _____ not passed _____ the Thesis defense in partial Fulfillment of the requirements of the degree of ~~MA~~/MS in Computer Science

Comments/Required Changes to Thesis due on (dd/mm/yy) ____________

→ long list provided to the student

Advisor's report on completion of above Thesis conditions: changes completed

Changes Approved by Thesis Advisor: N. Mansour Signature: [illegible] 13/1/2012

Date: 13/01/2012 Acknowledged by [redacted]
(Dean of Graduate Studies/School of .Arts & Sc.)

cc: Department Chair
Thesis Advisor
Registrar
Dean of Graduate Studies/School Dean

Date of the Thesis defense public announcement (dd/mm/yy) 14/9/2011

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

LEBANESE AMERICAN UNIVERSITY

School of Arts and Sciences

Thesis Approval Form

Student Name:Hratch Zeiutnlian I.D. #: 200402622

Thesis Title

Metaheuristic Algorithm for Testing Web 2.0 Applications

Program	:	Master of Science in Computer Science
Department	:	Computer Science and Mathematics
School	:	School of Arts and Sciences
Approved by	:	

Thesis Advisor:	Dr. Nashat Mansour	Signature : [redacted]
Member :	Dr. Sanaa Sharafeddine	Signature :
Member :	Dr. Abbas Tarhini	Signature :

Date : 13/01/2012

THESIS COPYRIGHT RELEASE FORM

Name: Hratch Zeitunlian

Signature: [redacted] Date: 13/01/2012

PLAGIARISM POLICY COMPLIANCE STATEMENT

I certify that I have read and understood LAU's Plagiarism Policy. I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me.

This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: Hratch Zeitunlian

Signature: [redacted] Date: 13/01/2011

ACKNOWLEDGMENTS

This research would not have been possible without the help and assistance of many persons.

First I would like to express my gratitude to my supervisor Dr. Nashaat Mansour.

I am also deeply grateful to Dr. Abbas Tarhini for sharing his expertise in the field of testing and metahuristics. Thanks go also to Dr. Sanaa Sharafeddine, for being on my thesis committee.

Finally, special thanks go also to my friends and family for their long support.

To my parents

Metaheuristic Algorithm for Testing Web 2.0 Applications

Hratch Michel Zeitunlian

Abstract

This thesis presents a new web application testing technique that addresses the complexity of WEB 2.0 Applications. Although significant work has been reported on state-based testing, not much of this work has addressed the particularities of modern web applications. In this thesis, we model the dynamic features of WEB 2.0 application by associating features or web pages with states; state transition diagrams are based on semantically interacting events responsible for state transitions. Test cases are generated as sequences of semantically interacting events and optimized using a metaheuristic algorithm. The metaheuristic is a simulated annealing algorithm that is based on concepts derived from physics. It is iterative and uses probabilistic search with the goal of minimizing an objective function. We formulate an objective function that is based on the capability of test cases to provide high coverage of events, high diversity of events covered, and definite continuity of events. The experimental results show that the proposed simultaneous-operation simulated annealing algorithm gives better results than an incremental version of the metaheuristic and significantly better than a greedy algorithm. We note that the proposed technique accounts for new features of web applications such as significance weights that can be assigned to events leading to significant features or pages, which ensures that test cases will be generated to cover these features.

Keywords: metaheuristics, search based software engineering, simulated annealing, software testing, state-based testing, test case generation, web applications

TABLE OF CONTENTS

TABLE OF TABLES

TABLE OF FIGURES

CHAPTER ONE

Introduction

Testing is an essential part of software development cycle. It is used to detect errors, and to ensure the quality of the software. Regardless of which software development model used, development process includes a testing stage at different points.

With traditional software, which usually follows the waterfall model, testing is applied when the coding process has been completed. However web applications differ from traditional software development where they follow the agile software development model, which has shorter development time. Because of the short development time web applications usually lack necessary documents and become user-centric feedback guided. This makes testing and maintaining web applications a more complex task.

During the past Decade radical changes were introduced in the development of web application. These changes pushed forward the conceptual mutation of the web, where the web is approached as a platform, and software applications are built upon. Thus the emergence of new generation of web applications and web system known as web 2.0. Web 2.0 applications are based on highly dynamic web pages, build around AJAX technologies, which through the asynchronous server calls, enables users to interact and affect the business logic on the servers.

Ajax technology created an umbrella under which the web 2.0 applications facilitated a high level of user interaction and web page dynamics. Examples are Google Maps, Gmail, Google Documents, Facebook, Yahoo mail and more.

The Dynamic features of web 2.0 impose additional complexity to the already hard task of web application testing. The complexity is found in the absence of traditional navigation paths. A complete web 2.0 web application can be made from a single page whose content and functionalities change by asynchronous server calls raised by the user interaction with the application, which changes the state in the client site resulting into a dynamic DOM. It is not possible to walk through the different states of the dynamic page since there is no unique URI assigned to a specific variant of the Dynamic Page unlike in traditional web applications where we have an explicit and unique URI for each Web page and each variant of a dynamic web page.

To test Web 2.0 applications and cover the dynamic aspects of the web 2.0; widgets, third party applications that can be executed within WebPages, Web parts, Portliest and hypermedia, we suggest a state based testing strategy that will dynamically generate a finite state machine from a web application by extracting semantically interacting events [1] that produce state changed in the user interface. From the inferred graph test cases will be generated having a sequence of events. Empirical results show that the longer the test case sequence, the more fault detection capability. However; generating test case from the finite state machine can lead to a very large test suite which can limit the usefulness of the method. Marchetto et al and Paolo Tonella suggested search based approach to generate long sequences of events while keeping the test suite size reasonably large using a greedy hill climb algorithm. The problem with the greedy algorithm is that the solution will be a local optimum rather than being a global optimum.

The objective of the research is to come with a more effective state based testing for a web 2.0 application that will cover all dynamic features of web 2.0 Application. In this we will be using Heuristics and not Graph algorithms, why not graph algorithms because our problem is to come up with a good sub optimal test suite with test cases that will reduce the test suite size while keeping the fault revealing power of the test suite. Whereas traditional path coverage techniques (Node Coverage or transition coverage) will generate a very big number of test cases whose number will increase exponentially ,as the maximum sequence length of the in a test cases increase.

To accomplish our strategy we formulate our approach around simulated annealing. The metahuristic algorithm adapts the dynamic nature of web 2.0 application whose test cases require back and forth state traversal, such a traversal between the states generates loop patterns included in the events sequences. Graph algorithms do not handle loops smoothly. Graph algorithms that will generate a large set of combinations and possibilities which doesn’t server our aim of coming up with the reduced test suite with best configuration of test cases. The simulated annealing will manage a best configuration of a fixed size test suite that suffices the desired test suite compositions characteristics which in our case, are test case diversity, lack of discontinuity in event sequences and event coverage which reflects the functionalities.

The rest of the work is organized as follows. In Chapter 2, we give a brief description of the testing problem and the objective of our research. In addition to background information and previous work done on testing web applications. Chapter 3 will describe our proposed solution, the simultaneous-operation simulated annealing, incremental simulated annealing and the greedy algorithms, while the experimental results are in Chapter 5; finally we give our conclusion in Chapter 6.

CHAPTER TWO

Literature Review

2.1 Background

In the effort to reduce application testing costs and improve software quality a lot of work has been done on automating testing techniques. One of the approaches used to automate test case generation is based on state machine model or even flow model [13].

State based testing is ideal when dealing with sequences of events. In some cases, the sequences of events can be potentially infinite, which of course exceeds testing capabilities, thus the need to come up with design technique that allows handling sequences of random lengths .

State based testing model has proved to be a successful approach specially when dealing with GUI testing. However the approach is considered to be resource-intensive specially while generating the model due to the significant manual intervention needed.

To improve the cost effectiveness of the method and reducing the number of possibilities the state based testing is extended to be formulated on a feedback strategy [13].

When using state machines to model a web 2.0 application states represent the user interfaces and the state transitions represent the events triggering the transition. A test case is a sequence of events that correspond for a path in the FSM.

FSM representation of Web 2.0 Application like all modern application have scaling problem because of the large number of candidate states and transitional events. Several suggestions were proposed by researchers to handle the scalability issue based on path search algorithms.

Several variants of FSMs have also been used for testing. The mutations are driven from the main aim to reduce the total number of states, and algorithms traverse these machine models to generate sequences of events as test cases.

These techniques require an initial test suite to be created, either manually or automatically, to be executed and evaluated. The feedback resulting from the evaluation is used to permute the initial configuration to automatically enhance or generate new test cases. The evaluation of feedback strategy is formulated mainly around the optimization algorithm used to target a specific goal. The targeted goal can be one of many however usually they are code coverage or state coverage or diversity to improve the overall performance of the test suite [14].

Alesandro Marchento and Paolo Tonella in their research on Testing Ajax enabled web applications prove the effectiveness of state based testing in finding faults [1]. In their initial work they generate a test suite of all paths of the same length K test cases derived from Finite State Machine representing the web application. Unfortunately empirical studies show that the effectiveness of this method however they highlight a major drawback presented in the very large test suite that may limit the usefulness of the test suite. To improve the cost effectiveness of their method Marchento And Tonella investigate test suite reduction using a search algorithm based on Hill-Climbing to deal with the problem of generating test cases out of long sequences of events on the same time keeping the test suite size reasonably small without degrading the fault revealing power of the exhaustively generated test suite.

2.2 Previous Work

2.2.1 State Based Testing

Extended Finite State machine (EFSM) is another model which is largely used for software testing. The EFSM model extends the classic FSM model with input and output parameters, context variables, and predicates. It is a remedy for the state explosion problem which FSM models face by inferring huge number of states.

In contrast to the Finite State Model which can be used to generate test suites that guarantee complete fault coverage. Or a complete test suite within the bounds to detect mutant Finite state Machines with in a predefined number of states. An EFSM can often be viewed as a compressed notation of an FSM. It is possible to unfold it into a pure FSM by expanding the values of the parameters, assuming that all the domains are finite. However this expansion should be carefully designed so as not to fall into the same trap of state explosion.

A.Petrenko and S. Boroday [15] call the state of unfolded EFSM as "configuration" and investigate the problem of constructing a configuration of sequences from an EFSM model , specifically when unfolded EFSM states result in generation of sequences that are different from sequences obtained from the initial configurations or at least they are not in the maximal subset. The authors generalize the problem into a search problem generating configurations sets. They demonstrate how the problem can be tackled and EFSM reduced so that existing testing methods that rely on FSM can handle the configurations as input. They present a theoretical framework for determining configuration-confirming sequences. Based on EFSMs .Moreover they elaborate on different derivation strategies.

The authors argue that the proposed approach of confirming sequence generation can be used to improve any existing test derivation tool that typically uses a model checker mainly to derive executable preambles and post ambles.

Tarhini, Fouchal, and Mansour presented a safe regression testing technique, for web service based applications [7]. In their work they target the challenges of the distributed system over heterogeneous networks in addition to availability and reliability of web service based systems. Being volatile systems prone to periodical changes and modification of web services, Web service based applications require to be tested fully, to guarantee coherence with the structural changes. Thus regression testing needed to select test cases from the original test suite generate during the initial development phase, and generate new ones to test the modification and newly added modules.

In their work the authors propose a regression testing method that is safe, by retesting the entire web system upon any modification. They base their technique on modeling the web application as a two level abstract model, and generating test cases sequences and test histories for the initial development. The test case generation is performed in exhaustive method that explores the entire space thus it inherits the exponential explosion of test case generation. The technique proposed lacks selective testing strategy to avoid the generation of large test suites.

Memon and Pollack worked on AI planning has to manage the state-space explosion by eliminating the need for explicit states [16]. In their work the GUI description is manually created by a tester; in the form of planning operators, which model the preconditions and post-conditions of each GUI event. The planner automatically generates test cases using pairs of initial and destination transitional states. The authors prove the efficiency of the system and suggest to be integrated with all FSM based modeling techniques.

Recently Alessandro Marchetto and Paolo Tonella worked on Web testing based on State Based testing for AJAX enabled web applications [1], to shed light on faults introduced by the asynchronous calls between the client and server.

The technique is based on inferring a finite state machine out of the Ajax application. State based testing is originally defined for event driven object oriented programs and lately used in GUI testing [1]. Due to the similarity between GUI applications and Web application specially AJAX application, that are built around a dynamic DOM structure manipulated by events Alessandro Marchetto and Paolo Tonella represent the web application by a Finite State Machine which depicts the state transitions and the events responsible for those transitions.

In their work the authors avoid using state based techniques, such as transition coverage or state coverage since such strategies have the potential of deriving large number of test cases. And they propose test suite reduction by adopting state based testing approach based on the notion of semantically interacting events.

In the tests performed Tonella and Marchetto show the test suite size reduction ratio between the non-semantic sequences and semantically interacting sequences. The size reduction obtained is between 78% -87% across different test case lengths. Moreover the results reveal an exponential growth of the number of test cases with the increase of the event sequence length. Test cases with sequences between 5 – 11 become very large thus resulting in an unmanageable test suite.

However, the technique proves its effectiveness in finding faults. Where relatively short length test cases composed of a sequence of four semantically events were able to detect 90% of the injected faults. On the other hand the technique inherited the problem of generating very large number of test cases especially with a long sequence of events in a test case thus limiting the usefulness of testing and test suite reduction method.

2.2.2 Search Based Testing

In a paper published later Alessandro Marchetto and Paolo Tonella address test suite reduction solution by generating controlled sequences of events and propose a heuristic, a greedy algorithm Hill Climb [2] to generate test cases out of short event sequences while keeping the test case number reasonably small in the suite to preserve the fault revealing power comparable to that of exhaustive test suite.

The Hill Climbing algorithm described in their work is a search algorithm that is guided by an objective function. It is used to evaluate an initial test suite and perturb member test cases. The perturbation is guided by the objective function, thus if the changes improve the fitness of the configuration it is accepted. At the end the obtained test suite will be an optimized test suite. However hill climbing will result in a local optimal solution instead of a global one.

Perturbations are done by concatenating a semantically interacting event at the end of an existing test case [2].

The authors base their fitness function on the notion of test suite diversity which is calculated by the frequency of each event covered in the FSM. Moreover, they experiment their algorithm by using different measures as fitness. EDiv which represents diversity based on the execution frequency of each event. PDiv test suite diversity based on the execution frequency of a pair of semantically interacting events. TCov which is the test suite diversity based of the FSM coverage.

Experimental results show the effectiveness of the Hill Climb algorithm in reducing the size of the test suite. The comparison between the different variants of the algorithm using the different fitness measures show that Edge Diversity yields to better results than the others. The test suite obtained via Edge Diversity maintains a high level of fault revealing capability.

CHAPTER THREE

Web Testing Problem and Research Objective

3.1 Web 2.0 Testing Problem:

With the shift in technology Web Applications are no longer static pages, but light client applications, that offer more features than the traditional web applications used to. With the development of the Web Industry, Web 2.0 Applications increasingly play an important role on daily activities and deal with increasingly sensitive data [3].

The correctness of a web application's User Interface is a good reference ensuring the correct operation of the overall web application. Comprehensive testing is a way to insure the correctness of the user interface. UI testing requires that test cases to be composed of UI sequence of events that invoke UI State changes when executed. The most common technique used to test UI is the capture and replay method which reacquire human intervention and test cases are generated manually by the user recording certain scenarios.

An important factor of cost effectiveness is optimizing the test suite, specifically the composition of test cases in the test suite. Test suite composition and test suite size have been hot research topics for a long time thus attracted a lot of debates around it. While some researchers suggest large test cases as small test suite units arguing that a large, not overly complicated test case, is more efficient than simple test cases. Others suggest large test suites with small test suites are more effective, since small test cases result in fewer cascading errors and large test suites are useful to expose system failures. Although those arguments refer to the size of the test case and test suite however it is just a reflection of a more complex issue, the issue of test case composition. Experimental work show that test suites containing test cases of varying lengths perform better. However the sequence lengths should be controlled by logic. Test suites composed of many small test cases can be none effective when testing Web 2.0 Applications that have complex state dependencies.

We believe that for most software systems and specially WEB 2.0 application test suites most have at least some level having varying length test cases is necessary.

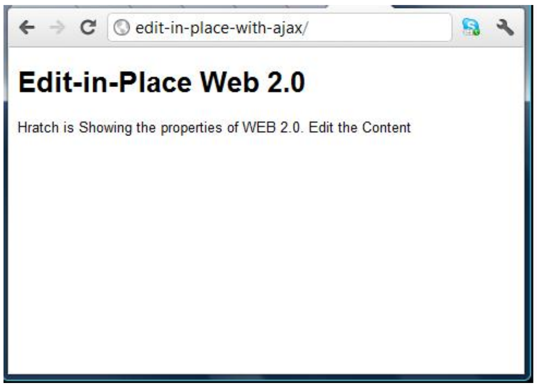
Existing web testing techniques lack the capability to handle the features of a WEB 2.0 Application. Unlike traditional Web Applications, Web 2.0 applications are often single page applications thus they lack traditional navigation paths. User interaction with the interface changes the structure of the content build around the DOM. Navigation in traditional Web applications is composed of hyperlinks where as in a Web 2.0 application every HTML element is able to produce navigation or a State change since an event can be attached to it at runtime.

Thus traditional white box testing techniques like Code Coverage [13] used to test web applications will fail in testing WEB 2.0 applications efficiently. Code Coverage technique is based on static analysis of the source code. Statically analyzing Web 2.0 code does not reveal request call backs. Thus the code coverage model that represents the web application statements executed as nodes and the edges as control transfers will not cover functionalities provided by asynchronous call backs. And functionalities and events added at runtime.

Such functionalities are heavily employed in Web 2.0 light client applications where Java script code is used to modify both Structure and content of HTML elements such as <Div>,<P>.

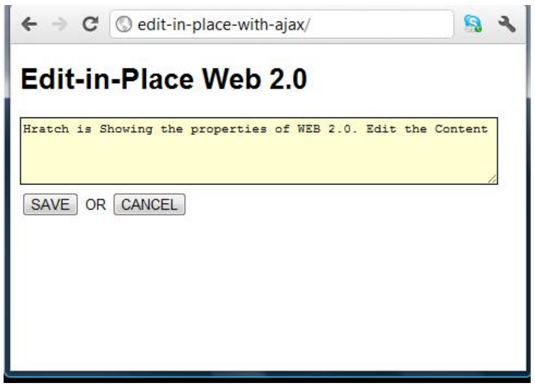
To illustrate dynamic capabilities WEB 2.0 application here is a simple example is of inline editing a text area. Figure 3.1 show the html code of the WEB 2.0 Page. The HTML element to make it editable is the element <P>with an attribute ID= “hmz”. The page functionality is as follows:

1. onMouseOver Highlight the text in <p> .
2. onMouseOut hide the highlight .
3. on click, hide the area to be edited and replace with the <p> with a <textarea> and <input> elements .
4. Remove all of the above if the user cancels the operation
5. on Save button click, execute an Ajax POST and show that busy page state animation.
6. on Ajax callback , update the page with the modified content.



Edit-in-Place Web 2.0

Hratch is Showing the properties of WEB 2.0. Edit the Content



Edit-in-Place Web 2.0

```
Hratch is Showing the properties of WEB 2.0. Edit the Content
```

Figure 3.1- HTML Code of Edit-in-Place Web 2.0 Interface before and after clicking the text area.

```
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
   <title>Edit-in-Place Web 2.0</title>
</head>
<body>
    <h1>Edit-in-place</h1>
    <p id="hmz"> Hratch is Showing the properties of WEB 2.0. Edit the content </p>
 </body>
 </html>
```

Figure 3.2- HTML Code of Edit-in-Place Web 2.0

```
<body>
   <h1>Edit-in-place</h1>
    <p id="hmz"> Hratch is Showing the properties of WEB 2.0. Edit the content </p>
   <div id="desc_editor">
      <textarea id=" hmz _edit" name=" hmz " rows="4" cols="60">Hratch is Showing the properties
    WEB 2.0. Edit the Content </textarea><div>
         <input id="desc_save" type="button" value="SAVE">
         OR
         <input id="desc_cancel" type="button" value="CANCEL"></div>
   </div>
</body>
```

Figure 3.3- HTML Code of Edit-in-Place Web 2.0 after click.

The java script functions in figure 3.4 are responsible for attaching “click” , “mouseover” and “mouseout” events to the candidate HTML element <P>. Moreover the click event is assigned an event handler “edit”, the execution of the “edit” event at runtime will insert into the DOM two buttons Save and Cancel. The buttons in their turn are attaching to the click events that trigger the function saveChanges and cleanChanges.

```
Event.observe(window, ‘load’, init, false);
 function init(){
    makeEditable(‘hmz’);
 }

 function makeEditable(id){
    Event.observe(id, ‘click’, function(){edit($(id))}, false);
    Event.observe(id, ‘mouseover’, function(){showAsEditable($(id))}, false);
    Event.observe(id, ‘mouseout’, function(){showAsEditable($(id), true)}, false);
 }

 function showAsEditable(obj, clear){
    if (!clear){
        Element.addClassName(obj, ‘editable’);
    }else{
        Element.removeClassName(obj, ‘editable’);
    }
 }
```

Figure 3.4 Javascript functions attaching events to HTML element

```
function edit(obj){

    Element.hide(obj);

    var textarea =’‘;

    var button = ‘ OR‘;

    new Insertion.After(obj, textarea+button);

    Event.observe(obj.id+’_save’, ‘click’, function(){saveChanges(obj)}, false);

    Event.observe(obj.id+’_cancel’, ‘click’, function(){cleanUp(obj)}, false);

}
```

Figure 3.5- JavaScript Add two buttons save and cancel.

```
function saveChanges(obj){
    var new_content = escape($F(obj.id+'_edit'));


    obj.innerHTML = "Saving…";
    cleanUp(obj, true);

    var success = function(t){editComplete(t, obj);}
    var failure = function(t){editFailed(t, obj);}

    var url = 'edit.php';
    var pars = 'id=' + obj.id + '&content=' + new_content;
    var myAjax = new Ajax.Request(url, {method:'post',
        postBody:pars, onSuccess:success, onFailure:failure});
}

function editComplete(t, obj){
    obj.innerHTML = t.responseText;
    showAsEditable(obj, true);
}

function editFailed(t, obj){
    obj.innerHTML = 'Sorry, the update failed.';
    cleanUp(obj);
}
```

Figure 3.6- JavaScript function call AJAX request to update the DOM.

The above example shows the level of complexity added by the new technologies used to develop web 2.0 applications where each GUI element can force state changes at runtime. More over the newly derived states will contain more clickable elements that expose Web application functionalities previously hidden from the user.

3.2 Research Objective

The objective of this thesis research is to tackle the drawbacks of the optimization solution presented by Marchento and Tonella, in addition to modifying the State Based testing technique [1] to handle the dynamic feature of Web 2.0 applications. Although the empirical result show the effectiveness of their method however their optimization is formulated around an aggressive hill-climbing algorithm [2] whose solution is in the local minimum and not a global optimum. We propose simultaneous-operation simulated annealing algorithm and gives better results than an incremental version of the metaheuristic as an alternative to the greedy algorithm, that will take the solution out of the local minima and result in a good sub optimal test suite that will reduce the size of the test suite without losing it power in detecting faults.

We formulate our optimization algorithm around methahuristics and not on graph algorithms, since simulated annealing will adapt gracefully to the nature of Web 2.0 applications that we are applying our testing method on. Graph algorithms although can guarantee graph coverage however; However without prioritization. Thus using graph coverage algorithms will result in huge number of possibilities, especially in the presence of loops, which makes the selection and generation of test cases of a test suite almost untraceable. Moreover Graph algorithms do not account for a combination of factors such as diversity, coverage in addition to continuity.

In addition to the fact that when dealing with long sequences the graph coverage possibilities increase exponentially. Thus the need to control sequence lengths and to come up with a sub optimal solution of a fix sized test suite that will represent the best candidate test cases to be executed.

Unlike previous work done on State based testing [1][2] for web applications where inferring the state graph required a significant amount of manual interference, as well as user interaction logs and possible input from outcomes of previous black box tests. Our proposed method will fully automate the generation of the finite state machine without the need for functionality trance. This level of automation will be reached by detecting clickable elements in the client DOM and automatically executing it. The auto executing of clickable events enable us to cover all provided functionalities by the web application even those functionalities that are never or rarely used by the user or used only from a particular state and not from within different states. In addition to the auto detection and execution of events our proposed strategy allows us to differentiate core functionalities from add-ons and third party code this can be achieved by adding an attribute to HTML element of the core component distinguishing them from similar elements introduced by an add-on or third party code. Thus the possibility of assigning importance weights for events covered by core components.

CHAPTER FOUR

Proposed Solution /Methodology

4.1 Motivation

Web 2.0 applications are constructed around highly dynamic web pages. The structure of these pages is constructed over a Dynamic DOM that is manipulated by the asynchronous server messages initiated by the client. To test Web 2.0 Applications and its dynamic features we feed the system with a Finite Sate Machine which represents the DOM states and the events that are responsible of the transitions [1][2]. Using the Finite State Machine test cases can be generated via different techniques however previously defined strategies usually suffer some drawbacks by generating high number or ending in a local optimal solution.

A greedy hill-climbing algorithm [2] was used to generate test sequences with best set of semantically interacting events; however, such algorithms will gradually get stuck in local minima. In this thesis, we chose simulated annealing strategy to generate test sequences because it allows uphill moves which will forces the solution to jump out of a local minima and fall into a more promising downhill in a controlled way.

The reason to select a metahuristic algorithm to solve our optimization problem lays in the nature of web 2.0 applications. To effectively test web applications and specifically WEB 2.0 application visiting the same state back and forth is essential, thus test cases generated to test WEB 2.0 applications should be capable of handling loops. Traditional Graph traversal algorithms do not handle loops efficiently thus they are not good candidates to generate test cases for Web 2.0 applications. More importantly our testing strategy focuses on test suite reduction and optimization. Graph coverage algorithms be it State coverage or transition coverage are capable of retrieving all independent paths of a graph but they lack the power to prioritize the output, Thus resulting into huge number of possible sequences. This number will increase exponentially as the maximum length of test cases increase which makes managing the test sequences unaffordable [2]. Our study our aim is to come with the best test suite with predefined size that containing best candidate set of test cases prioritizing event sequence continuity, test suite diversity and coverage.

To be able to compare our work with greedy hill-climbing algorithm [2] proposed by Alessandro Marchetto and Paolo Tonella which is greedy incremental algorithm where events are added on sequences to generate longer sequence if the addition of the new test case improves the test suite configuration it is accepted else rejected. We formulate an incremental simulated annealing algorithm that at will generate the test suite incrementally by adding test case after each Simulated Annealing cycle. The added test case will be presenting the best configuration given previous decision, added test cases, into the test suite.

In contrast to the incremental simulated annealing, simultaneous operations simulated annealing algorithm is formulated to fully utilize the power of simulated annealing. The algorithms will handle the entire test suite composition and will be perturbing the test cases simultaneously to reach an optimum configuration.

In addition to the two the simulated annealing algorithms we formulate a greedy algorithm that will be searching for an optimized configuration of the test suite in the neighborhood solution by perturbing the test suite and checking of improvement in the fitness values.

In addition to the test suite reduction technique, in this thesis we propose an automated method to infer a finite state machine out of the States of Web 2.0 application. Unlike the proposed method by Alessandro Marchetto and Paolo Tonella in their state based testing work [1][2] where inferring a finite state machine requires   manual work to refer to traces and some level of functionality testing before proceeding with the graph generation. The method we define will allow automatic state generation by detecting clickable events responsible for state transitions and executing them automatically.

4.2 Graph Modeling

Extracting a state graph form a Web 2.0 application is not a direct and simple task. The main challenge is the absence of traditional navigational paths. This is because in Web 2.0 there is no unique URI assignment to a specific variant of the Dynamic Page, unlike traditional web applications where each web page state in the browser has an explicit URI assigned to it [2]. Moreover, an entire Web 2.0 application can be created from a single web page where   User Interface (UI) is determined dynamically through changes in the DOM initiated by user interaction through asynchronous server calls. Further, Web 2.0 application may contain third party HTML Units, User shared data, widgets and media content that are added to the application simultaneously during execution. To overcome the above mentioned challenges our testing mechanism will reconstruct the user interface states, and generate static pages having Navigation Paths each with unique URL. These Static pages will be used to conduct State-Based testing [1].

To achieve the static-like pages we need a tool that will execute client side code, and identify clickable elements which may change the state HTML/ DOM within the browser[1][2]. From these states changes we will build our state graph that captures the states of the user interface, and the possible transitions between the states.

4.2.1 The State Graph

Our Model must reveal all user interface state changes in Web 2.0 application. Thus the model must record all navigation paths/event of the DOM state changes. This can best be represented by a State Graph which is defined as follows.

Definition 2.1. A State Graph for a Web 2.0 site **A** is a 5 tuple <**r**,**V**,**C**, **E,W**> where:

1. **r** is the root node representing the initial state after **A** has been fully loaded into the browser.
2. **V** is a set of vertices representing the states. Each **v** ϵ **V** represents a run-time state in **A**.

3. **C** is a set of clickable elements that enables the transition from one state to another.
4. **E** is a set of edges between vertices .Each $(\mathbf{v_1},\mathbf{v_2}) \in \mathbf{E}$ Represents a clickable $c \in \mathbf{C}$ connecting two states if and only if state $\mathbf{v_2}$ is reached by executing c in state $\mathbf{v_1}$.
5. W is the weight of e representing the importance of the event E

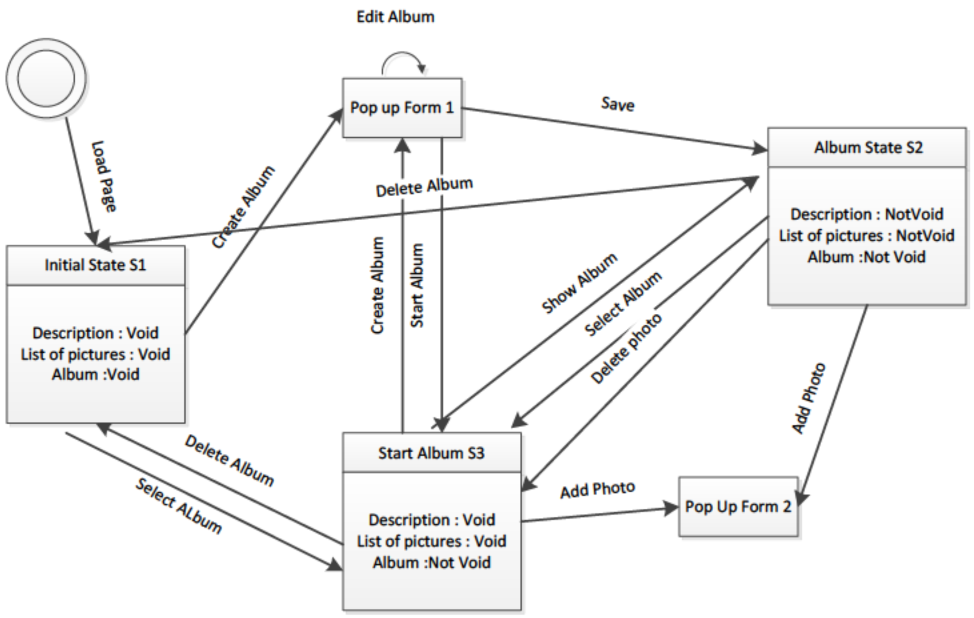


Figure 4.1- Example of a state graph model of a web application

Figure 4.1 depicts the visualization of the state graph for a simple Web 2.0 Application responsible for managing online photo albums. Its main functionalities are creating an album, delete existing album, select existing album, edit album, save album, add photo, delete photo, display album. Moreover the figure shows the three main states of the application S1 starting state, S2 where at least one photo is selected, and S3 an album is selected. It illustrates how the three different states can be reached.

Figure 4.2 Shows the HTML code of Online Album Management WEB 2.0 Application. For the sake of simplicity only some of the functionality is illustrated and code responsible for formatting and design is removed.

To infer the state graph of the online album management web application is loaded into the browser. Loading the webpage will generate state S1 in Figure 3.1. The state is characterized by having the entire HTML element set to Null or Empty Figure 4.3 shows the DOM tree after the page is loaded on the client web browser. Being the actual initial first state, S1 is added to the FSM. After DOM is loaded and Modified at the client side, it is preceded with the search of clickable element. The first clickable elements detected is “btnSelect” that triggers the event “select album” as show in HTML code in figure 4.2. The Button “btnSelect” is represented by an input element of type submit <input type = “submit” name= “btnSelect”> in the DOM. Thus the event “Select Album” is executed. The Execution of the “Select Album” event generates the state S3 in figure 3.1 represented in the by the DOM in Figure 3.4 which shows the Album selected and the album name element value filled as not empty. Initially the generated State S3 is analyzed and compared with the previously covered states. The obtained State being a new state will be added to the FSM and an edge marking the event “Select Album” will be added between the states marking the transitional event between the states S1 ,S3 as show in the Figure 3.1.

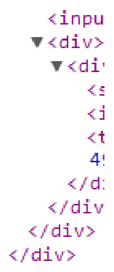
Next the clickable element <input type = “submit” name= “btnShowAlbum”> is detected and the event “Show Album” event is executed. The execution of the “show album” forces dynamic changes of DOM and a transition of a new state S2. The states S2 DOM representation in Figure 4.5 shows at least one photo thus an image element with non-empty image source and text element containing the description of the photos in addition to the name of the album. Comparing the State with previously obtained states it is marked as new and added to the FSM as depicted in Figure 3.1 which shows the directed edge “Show Album” connecting the states S3 to S2.

Continuing scanning for clickable events the button “btnDelete” is detected represented by the HTML element <input type = “submit” name= “btnDelete ”> and the event “delete album” is executed brining the state S2 into a new transition. Comparing the newly obtained state with previously generated states it is marked similar to the state S1 the initial state. Since the state is previously added in the FSM only the event “Delete Album” is added as an edge marking a transition between the states S2, S1 as depicted in Figure3.1.

Similarly all clickable elements will be detected and corresponding events executed, and the FSM generated covering all functionalities included by the WEB 2.0 Application.

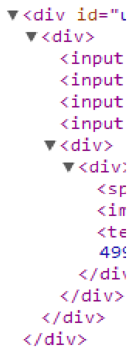
```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Ajax Photo Album Example</title>
</head>
<body>
    <form id="form1" runat="server">
      <asp:ScriptManager ID="ScriptManager1" runat="server">
                </asp:ScriptManager>
        <asp:UpdatePanel ID="upAjaxContent" runat="server">
            <ContentTemplate>
                <div>
                <asp:Button ID="btnSelect" runat="server" Text="Select"
                    onclick="Select Album" />
                     <asp:Button ID="btnDelete" runat="server" Text="Delete"
                    onclick="Delete Album" />

                    <asp:Button ID="btnEdit0" runat="server" onclick="edit album" Text="Edit" />

                    <asp:Button ID="btnShowAlbum" runat="server" onclick="Show Album"
                        Text="Show Album" />

                <div/>
                <div>
                 <asp:Label ID="lblAlbumName" runat="server"></asp:Label>
                <asp:Image ID="Image1" runat="server" Width="500px" />

                <asp:TextBox ID="txtDescription" runat="server" Height="75px"
                        TextMode="MultiLine" Width="499px"></asp:TextBox>
                </div>
            </ContentTemplate>
        </asp:UpdatePanel>

    </form>
</body>
</html>
```

Figure 4.2 Sample HML code of WEB 2.0 Album Management



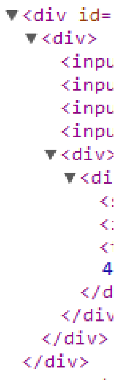
```
      <inpu
    ▼<div>
      ▼<di
          <s
          <i
          <t
          4!
        </d:
      </div
    </div>
  </div>
```

Figure 4.3 DOM of Initial State S1- no album selected



```
▼<div id="u
  ▼<div>
      <input
      <input
      <input
      <input
    ▼<div>
      ▼<div:
          <sp
          <im
          <te
          49!
        </di
      </div>
    </div>
  </div>
```

Figure 4.4 DOM of Start Album State S3 – An album is selected



```
▼<div id=
  ▼<div>
      <inpu
      <inpu
      <inpu
      <inpu
    ▼<div>
      ▼<di
          <:
          <:
          <·
          4
        </d
      </div
    </div>
  </div>
```

Figure 4.5 DOM of Album State S2 - At least one picture selected

When inferring the FSM two issues are to be considered while building. First we need to detect the event-driven elements; next, we need to identify the state changes. The State Graph is created incrementally; initially, the state graph contains only the root state. Additional states are appended to the graph as event-driven elements are traced / invoked in the application and state changes are analyzed.

4.2.2 Detecting Event-Driven Elements

Once an HTML page is loaded, we can access the HTML elements through the DOM. However, there is no direct way to detect the event driven elements; thus we need to introduce a candidate list of elements to be used as a reference. Candidate elements are elements that are invoked by different types of events like (Click, Doubleclick, MouseOver). For example, <div>, <input>, and <a> are candidate elements.

4.2.3 Detecting States and Inferring the FSM

As a candidate element is detected, we execute the event attached to that element. In order to determine whether the execution of the event results in state change, we compare the version of DOM-tree after firing the event and the DOM-tree version just before firing that event. If the execution of the event results in state change, we check whether the resulting state is already covered. To check State similarities we generate a hash code out of each loaded DOM-tree and compare it with the existing hash codes, if the state hasn't been covered previously it is added to the FSM, with an edge representing the transitional event connecting the two states. If the state is already covered, simply an edge will be added between the states.

4.2.4 Semantic Interactions

Definition 2.2. Semantically interacting events: Events $\mathbf{e_1}$ and $\mathbf{e_2}$ are interacting semantically if there exists as state $\mathbf{S_0}$ such that their execution in $\mathbf{S_0}$ does not commute, i.e., the following conditions hold:

$$\mathbf{S_0 \Rightarrow_{e1:e2} S_1 ; S_0 \Rightarrow_{e2:e1} S_2 ; S_1 <> S_2}$$

where $\mathbf{S_0}$ **,** $\mathbf{S_1}$**,** and $\mathbf{S_2}$ are any states in the state graph of the web application.

The notion of pair of semantically interacting events can be easily generalized to sequences [3].

Definition 2.3. Sequence of semantically interacting events: The event sequences $(e_1,\ldots e_n)$ is a sequence of semantically interacting event if every pair of events in the sequence is pair of semantically interacting events according to **Definition 2.2**.

4.3 Simultaneous-Operations Simulated Annealing

Simulated annealing is influenced by ideas from physics and is analogous to the physical annealing of a solid [11]. Annealing is used in metal to reach a state where the atoms are highly ordered. To reach this state material is heated and then cooled very slowly, allowing many atomic rearrangements till it comes to thermal equilibrium at each temperature drop.

The simulated annealing algorithm (SA) simulates the natural phenomenon by perturbations and search process in the solution space. The search is guided by an optimizing energy function. It starts with some badly unordered initial configurations at a high temperature and then gradually cooled down to a freezing point with a highly optimized best global solution [11]. In the following subsections, we describe how we generated test sequences of semantically interacting events using the simulated annealing algorithm; an outline of the SA algorithm is given in Figure 4.1.

In our work, we choose simulated annealing strategy to generate test sequences because it allows us to jump to a global optimal or sub optimal configuration, by controlled uphill moves that will allow more downhill moves thus pulling the solution out of local minima.

```
Initial configuration = Sequence of events from the state graph;
Determine initial temperature T(0);
Determine freezing temperature $T_f$ ;
while (T(i) > $T_f$ and not converged) do
          repeat several times
                    (multiple of the number and size of required test cases)
                    Generate_function();
         save_best_sofar();
         T(i) = $\theta$ * T(i);
endwhile

procedure Generate_function()
    perturb();
    if ($\Delta$OF1 $\leq$ 0 ) then
        update()                 /* accept */
    else
        if (random() < $e^{-\Delta OF1 / T(i)}$) then
                  update()       /* accept */
     else
        reject_purturbation();
```

Figure 4.6 Outline of the simultaneous-operation SA algorithm.

It is clear from the SA algorithm described in Figure 4.6 that SA strategy consists of four basic components.

1. Configuration
2. Perturbation
3. Energy Function
4. Cooling Schedule

4.3.1 Solution Representation

Our solution will be represented as a configuration C, which is implemented as an array of variable-length test cases. Each test case contains a maximum of K events derived from the State Flow Graph. The length of the array is K* N, where N is the maximum number of test cases required in the solution. To allow variable length of test cases, we will introduce fake edges into our set of valid events. These fake edges, called “No Edge”, will play the role of space holder in the array.

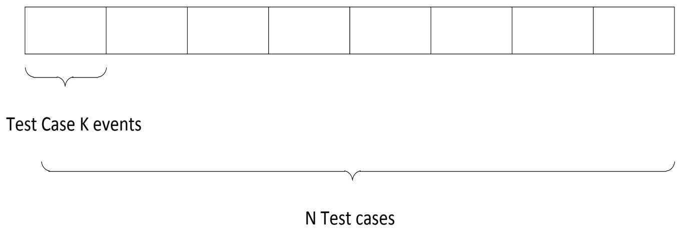


Figure 4.7 - Structure of the test Suite in simultaneous-operation SA.

4.3.2 The Metropolis step and feasibility

An iteration of the Metropolis [11] step, Generat.,e_function(), consists of a perturbation operation, an accept/reject criterion, and a thermal equilibrium criterion. Perturbation in our strategy is done randomly by selecting an event within a test case and substituting it with a randomly chosen event from the Events Set.

The acceptance criterion checks the change in **E** due to the perturbation. If the change decreases the objective function, the perturbation is accepted and C is updated. However, if the perturbation causes the objective function to increase, it is accepted only with a probability $e^{-\Delta OF1 / T(i)}$. The main advantage of this Monte Carlo algorithm is that the controlled uphill moves can prevent the system from being prematurely trapped in a bad local minimum-energy state. Note that for lower temperature values T(i), the probability of accepting uphill moves becomes smaller; at very low (near-freezing) temperatures, uphill moves are no longer accepted. The perturbation-acceptance step is repeated many times at every temperature after which thermal equilibrium is considered to be reached.

Perturbations can make C infeasible if they violate the definition of continuity. But, the formulation of the energy function **E** accounts for this infeasibility problem. The last term in **E** (DC) can be assigned a large weight, γ, so that infeasibility is severely penalized. Thus, infeasible test cases will be prevented at low temperatures.

4.3.3 Cooling schedule

The cooling schedule is determined by running a heuristic algorithm that deduces the starting and freezing temperatures with respect to the number of Uphill Jumps. The initial temperature T(0) is the temperature that yields a high initial acceptance probability of 0.93 for uphill moves. The freezing point is the temperature at which such a probability is very small (2-30), making uphill moves impossible and allowing only downhill moves. The cooling schedule used in this work is simple: $T(i+1) = \theta * T(i)$, with $\theta = 0.95$.

As the annealing algorithm searches the solution space, the best-so-far solution (with the smallest OF1) found is always saved. This guarantees that the output of the algorithm is the best solution it finds regardless of the temperature it terminates at. Convergence is then detected when the algorithm does not improve on the best-so-far solution for a number of temperatures, say 20, in the colder part of the annealing schedule.

4.3.4 Energy function

The Energy function measures how good the current configuration is. We based the energy function on three major weighted factors. The weights represent the importance of each factor. The three factors are Continuity, Diversity and Coverage.

Continuity:

When testing event based applications it is very important to test a continuous set of events. In fact, test cases with longer continuous sequences of events have higher capability of revealing faults. In our Simulated Annealing strategy we want to minimize the discontinuity of events in a test case. We calculate discontinuity by checking the events in every test case and incrementing the value by one whenever discontinuous events are found.

Diversity:

Diversity is an important factor which guarantees that test cases will cover events from the entire scope of the Web application and not just concentrate on events from a certain part, and therefore, we guarantee equally distributed events within the entire test suites. In this work we will be minimizing the Lack of Diversity by calculating the average frequency of events in the entire Test Suite. Thus, given a test suite S, composed of a set of test cases based on semantically interacting sequences of events, its Lack of diversity (Div) is computed as follows:

$$\text{Div} = \sqrt{\sum_{e \in Events} (F_e - F_{avg})^2}$$

where: **e** is an event that belongs to the set of events ***Events***, $\boldsymbol{F_e}$ is the execution frequency of event ***e***, and $\boldsymbol{F_{avg}}$ is the average frequency of event ***e*** computed over the entire test suite.

Weighted Coverage:

In Web 2.0 applications, end users and third parties can change the content of a web page dynamically by injecting HTML code or web widgets through their interaction with the site. Thus, some events would have higher importance than other events; accordingly, we may control or even limit some functionality from being included in our testing plan by allowing a measure of importance on events that are part of the original web application, compared to injected events or functionality into the web application. The importance of events is represented by pre-defined weights assigned to every event. Again we want to minimize the value of the unimportant events and this value is calculated by checking if an event is covered in the test suite and multiplying it with its importance or weight.

$$\text{WC} = \frac{\sum_{e \in Event} (W_e \; x \; C_e)}{\sum_{e \in Event} W_e}$$

Finally, the Energy function will be represented as:

$$\text{E} = \alpha \times \frac{\sum_{e \in Events} (W_e \times C_e)}{\sum_{e \in Events} W_e} + \beta \times \sqrt{\sum_{e \in Events} (F_e - F_{avg})^2} + \gamma \times DC$$

Where $\alpha, \beta,$ *and* γ are user-defined weights for weighted coverage, diversity, and discontinuity respectively.

Note that different values can be assigned to the weights in E. These weights are important for selecting test cases. They might be contradictory; that is, by increasing one of these weights, say α, the solution will improve in minimizing one factor (discontinuity) while it might increase the other factors. These weights will allow flexibility in using our proposed solution algorithms to suit the user's particular choices or requirements for different instances of the problem.

4.4 Incremental Simulated Annealing

Incremental Simulated Annealing is a mutation of the simultaneous-operation simulated annealing. While the latter deals with the full set of test cases in the test suite in a parallel manner. The incremental simulated annealing generates a single test case containing maximum of K events at each iteration and adds the test case to the final configuration of test suite. The algorithm makes use of the same Energy function. However; at the end of each iteration, the event frequency, coverage and diversity matrices are saved, to be used by the energy function on the next iteration.

```
While  testCases <  N  (Maximum number of test cases in the test
suite)
      repeat
      Increment  testCases
      Initial configuration = Sequence of  K events from the state
graph;
      Determine initial temperature T(0);
      Determine freezing temperature $T_f$ ;
          while (T(i) > $T_f$ and not converged) do
              repeat several times
                     (multiple of the number and size of required test cases)
                     Generate_function();
          save_best_sofar();
          T(i) = $\theta$ * T(i);
          Endwhile
   Save event  frequencies values
  Save  diversity  values

Endwhile

procedure Generate_function()
    perturb();
    if ($\Delta$OF1 $\leq$ 0 ) then
        update()                  /* accept */
    else
        if (random() < $e^{- \Delta OF1 / T(i)}$) then
                   update()       /* accept */
     else
        reject_purturbation();
```

Figure 4.8- Outline of the Incremental SA algorithm.

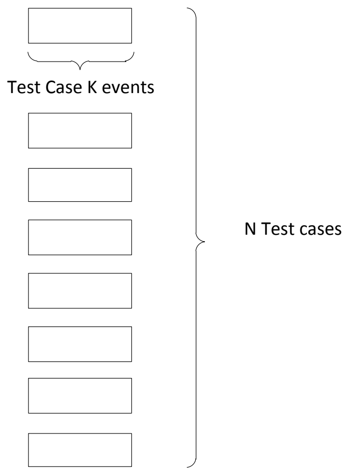


Figure 4.9- Structure of the Incremental SA algorithm.

4.5 Greedy Algorithm

The greedy algorithm is similar to the simultaneous-operation SA. What makes this algorithm greedy is that it neutralizes the Monte Carlo algorithm by accepting only the changes that decrease the energy of the objective function, and not allowing any Uphill moves. The Algorithm is guided by the same objective function and similar to the simultaneous SA it deals with the entire test suite instead of generating a single test case after each iteration.

```
Initial configuration = Sequence of events from the state graph;
Determine initial temperature T(0);
Determine freezing temperature $T_f$ ;
while (T(i) > $T_f$ and not converged) do
    repeat several times
        (multiple of the number and size of required test cases)
        Generate_function();
    save_best_sofar();
    T(i) = $\theta$ * T(i);
endwhile

procedure Generate_function()
  perturb();
  if ($\Delta$OF1 $\leq$ 0 ) then
    update()            /* accept */
  else
    reject_purturbation();
```

Figure 4.10- Outline of the Greedy algorithm.

CHAPTER FIVE
Experimental Results and Discussion

5.1 Experimental Procedure

To examine our research question we base our experiment on two different Finite State Machine models. First set of tests are done on an FSM representing a small web 2.0 application Web Application 1 (WebApp-1), constituting consisting of 36 States and having 86 events to generate an optimized test suite size of 50 test cases; each test case has a maximum of 8,9,10,18 and 20 events for each iteration. With the addition of 10% of the total events are fake events to introduce sequence discontinuities. The FSM is depicted in figure 5.2. The second set of test is applied on a bigger FSM representing a bigger Web Application (WebApp-2) consisting of 50 states and 270 events.

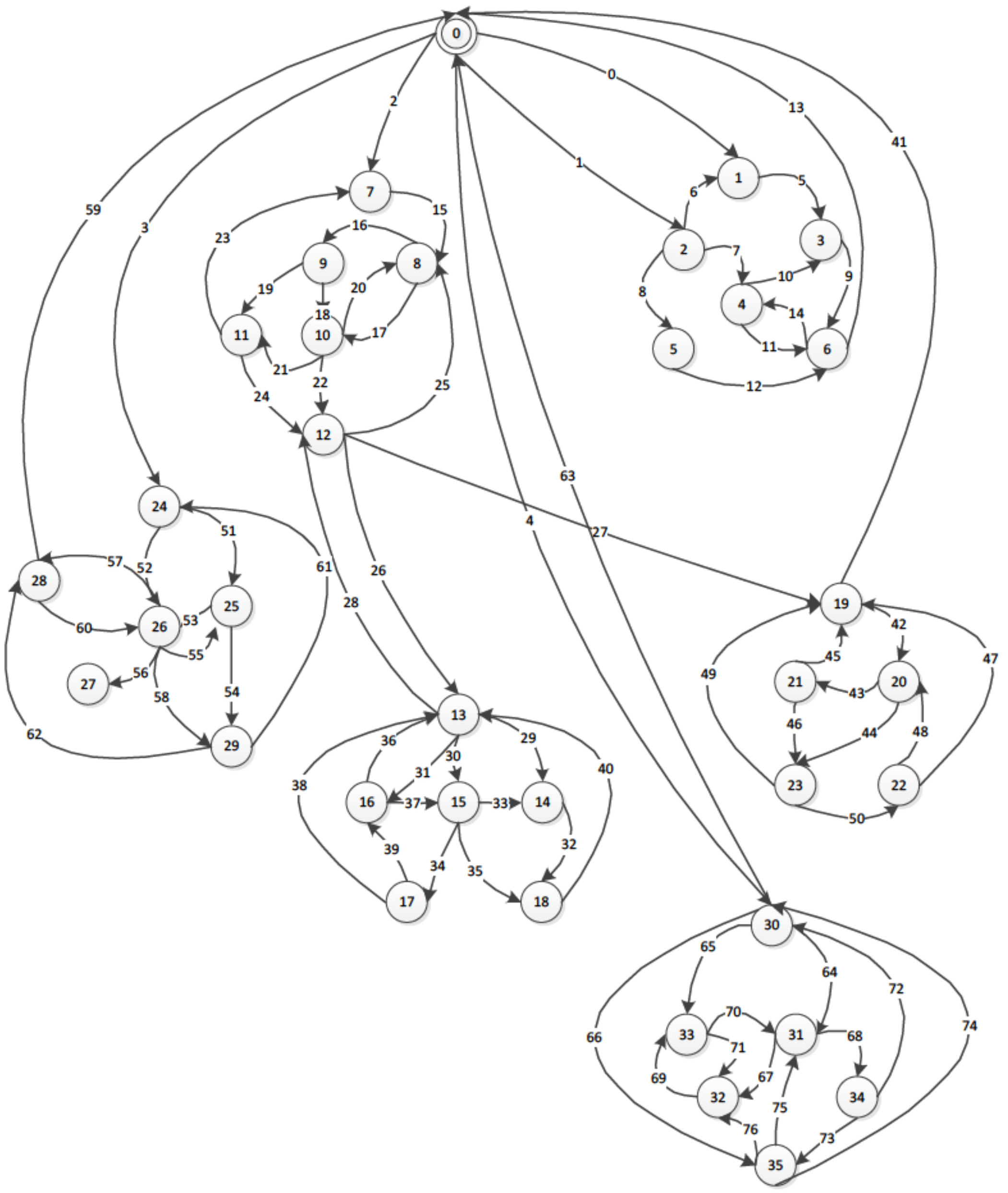


Figure 5.1 FSM of Web Application 1

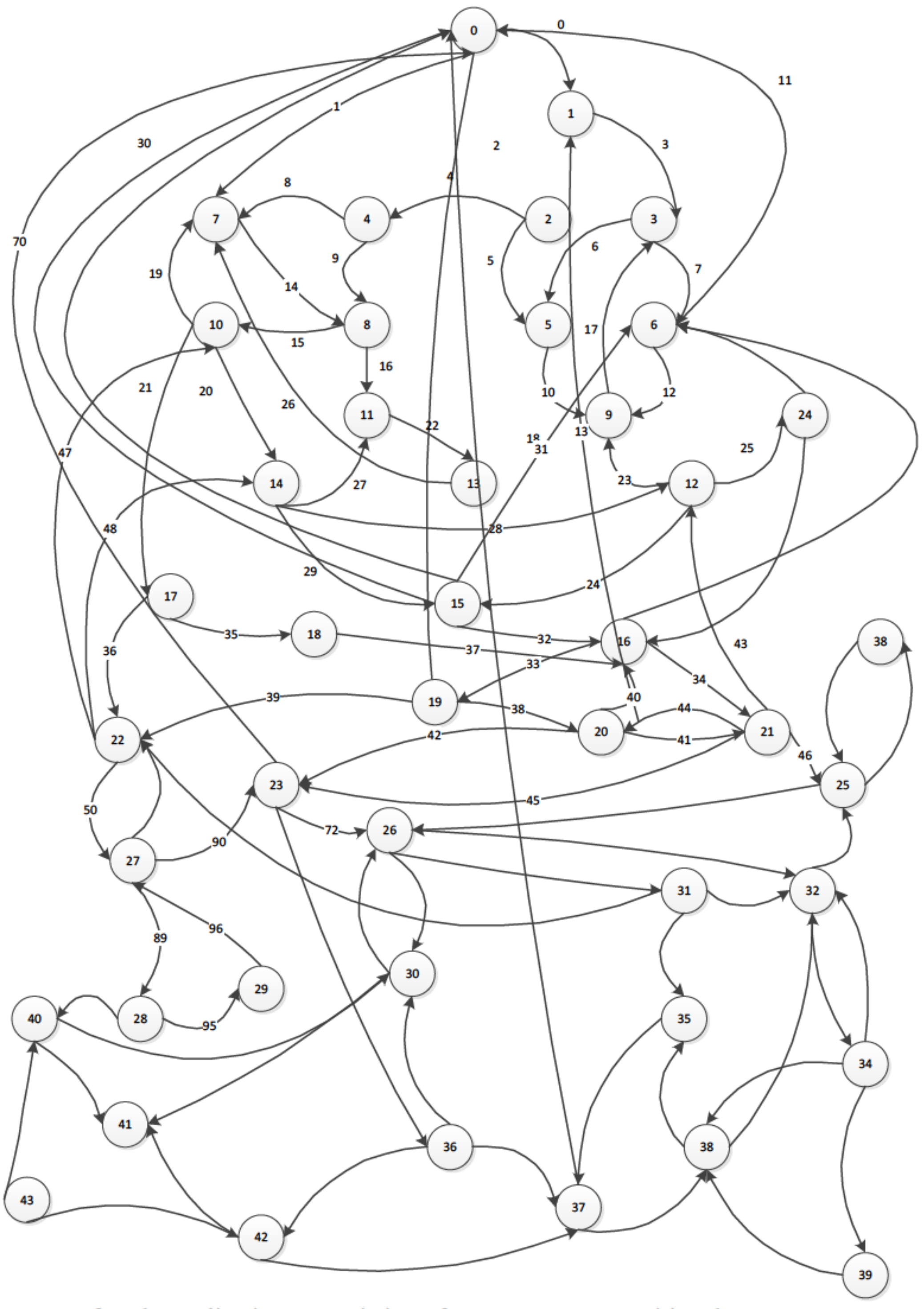


Figure 5.2 FSM of Web Application 2 consisting of 50 states 270 Transitional events.

5.2 Experimental result

Initially experiments are performed on web application 1, using Simultaneous-Operations Simulated Annealing algorithm. The algorithm successfully generates 50 test cases, consisting of continuous sequence of events.

Close examination of the results reveal that the algorithm successfully covers all the events in application, and more importantly it is able to generate a diversity suite that ensures, efficiently testing different parts of the WEB 2.0 application.

Figure 5.3 Shows some randomly selected subset of test cases from the Test suite of 50 test cases generated having maximum of 18 events in a test case. The nodes in the sequence represent the states and the arrows represent the event responsible for state transitions. While the first two test cases contain 18 events, the other three test cases have a shorter sequence length due the presence of fake events at the start/termination of test cases, specifically events 81, 82 and 83. Those fake events trim the test case length to a shorter sequence.

$$\begin{tikzcd}
28 \arrow[r, "60"] & 26 \arrow[r, "55"] & 25 \arrow[r, "54"] & 29 \arrow[r, "61"] & 24 \arrow[r, "51"] & 25 \arrow[r, "54"] & 29 \arrow[r, "61"] & 24 \arrow[r, "52"] & 26 \arrow[r, "57"] & 28 \arrow[r, "59"] & 0 \arrow[r, "1"] & 2 \arrow[r, "7"] & 4 \arrow[r, "11"] & 6 \arrow[r, "13"] & 0 \arrow[r, "4"] & 30 \arrow[r, "66"] & 35 \arrow[r, "75"] & 31 \arrow[r, "67"] & 32 \\
9 \arrow[r, "18"] & 10 \arrow[r, "22"] & 12 \arrow[r, "27"] & 19 \arrow[r, "41"] & 0 \arrow[r, "3"] & 24 \arrow[r, "52"] & 26 \arrow[r, "55"] & 25 \arrow[r, "53"] & 26 \arrow[r, "57"] & 28 \arrow[r, "59"] & 0 \arrow[r, "2"] & 7 \arrow[r, "15"] & 8 \arrow[r, "16"] & 9 \arrow[r, "19"] & 11 \arrow[r, "24"] & 12 \arrow[r, "26"] & 13 \arrow[r, "29"] & 14 \arrow[r, "32"] & 18 \\
21 \arrow[r, "45"] & 19 \arrow[r, "41"] & 0 \arrow[r, "1"] & 2 \arrow[r, "7"] & 4 \arrow[r, "11"] & 6 \arrow[r, "14"] & 4 \arrow[r, "10"] & 3 \arrow[r, "9"] & 6 \arrow[r, "14"] & 4 \arrow[r, "11"] & 6 \arrow[r, "13"] & 0 \arrow[r, "4"] & 30 \arrow[r, "64"] & 31 \arrow[r, "67"] & 32 \arrow[r, "69"] & 33 \arrow[r, "71"] & 32 \arrow[r, "83"] & F \arrow[r, "83"] & F \\
2 \arrow[r, "8"] & 5 \arrow[r, "12"] & 6 \arrow[r, "14"] & 4 \arrow[r, "11"] & 6 \arrow[r, "13"] & 0 \arrow[r, "0"] & 1 \arrow[r, "5"] & 3 \arrow[r, "9"] & 6 \arrow[r, "13"] & 0 \arrow[r, "3"] & 24 \arrow[r, "52"] & 26 \arrow[r, "55"] & 25 \arrow[r, "53"] & 26 \arrow[r, "58"] & 29 \arrow[r, "62"] & 28 \arrow[r, "60"] & 26 \arrow[r, "56"] & 27 \arrow[r, "84"] & F \\
F \arrow[r, "81"] & F \arrow[r, "82"] & 2 \arrow[r, "8"] & 5 \arrow[r, "12"] & 6 \arrow[r, "14"] & 4 \arrow[r, "10"] & 3 \arrow[r, "9"] & 6 \arrow[r, "13"] & 0 \arrow[r, "4"] & 30 \arrow[r, "66"] & 35 \arrow[r, "74"] & 30 \arrow[r, "63"] & 0 \arrow[r, "3"] & 24 \arrow[r, "52"] & 26 \arrow[r, "58"] & 29 \arrow[r, "61"] & 24 \arrow[r, "52"] & 26 \arrow[r, "56"] & 27
\end{tikzcd}$$

Figure 5.3 Some test cases derived by applying Simultaneous- Operations Simulated Annealing Algorithm to generate having maximum 18 events per test case for WebApp-1.

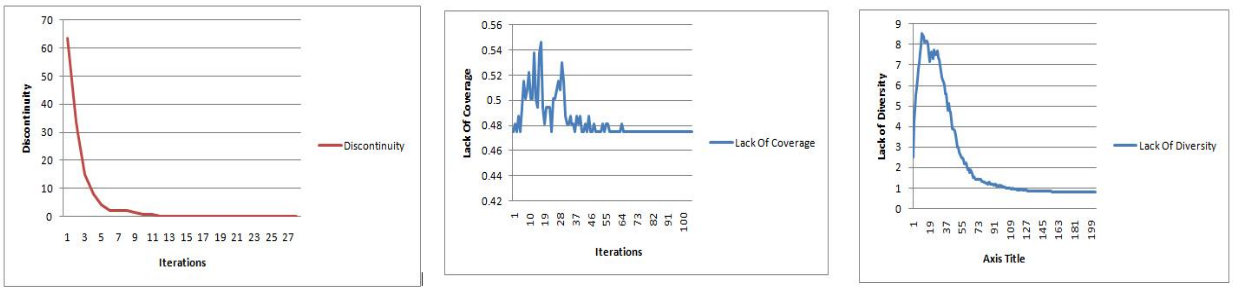


Figure 5.4 Discontinuity, lack of coverage, Lack of diversity value graphs of Simultaneous-Operations Simulated Annealing Algorithm to generate having maximum 18 events per test case, for WebApp-1.

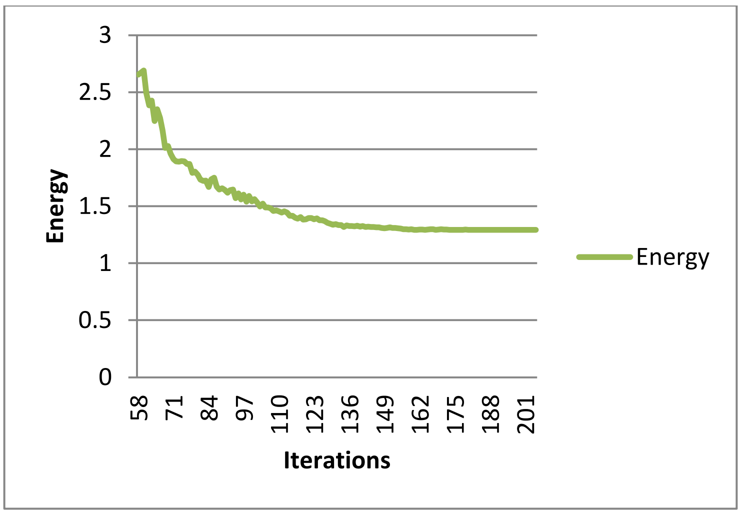


Figure 5.5 - Energy value of the Objective Simultaneous-Operations Simulated Annealing Algorithm K_{max}= 18 for WebApp-1.

The graph depicted in figure 5.5 shows the overall slow convergence of the energy value of the objective function.

The set of experiments is repeated using the Incremental Simulated Annealing algorithm. The results show that the Incremental simulated annealing algorithm successfully generates optimized test suites. However the performance of the Simultaneous-Operations Simulated Annealing is superior to that of the incremental algorithm. And this is because the Simultaneous- Operations Simulated Annealing algorithm yields to lower energy values the same test suite size with test cases of the same K_{max}. Table 5.1 presents the corresponding energy values for the test suites obtained with different values of K_{max} test cases using the different algorithms.

Repeating the experiment with the Greedy Algorithm results in an un optimized test suite. This is because the Greedy algorithms fails to generate continuous sequences of events in the test cases. The energy value converges fast within the initial little iteration; however no farther improvements are obtained. Figure 4.4 shows the graph of the energy values.

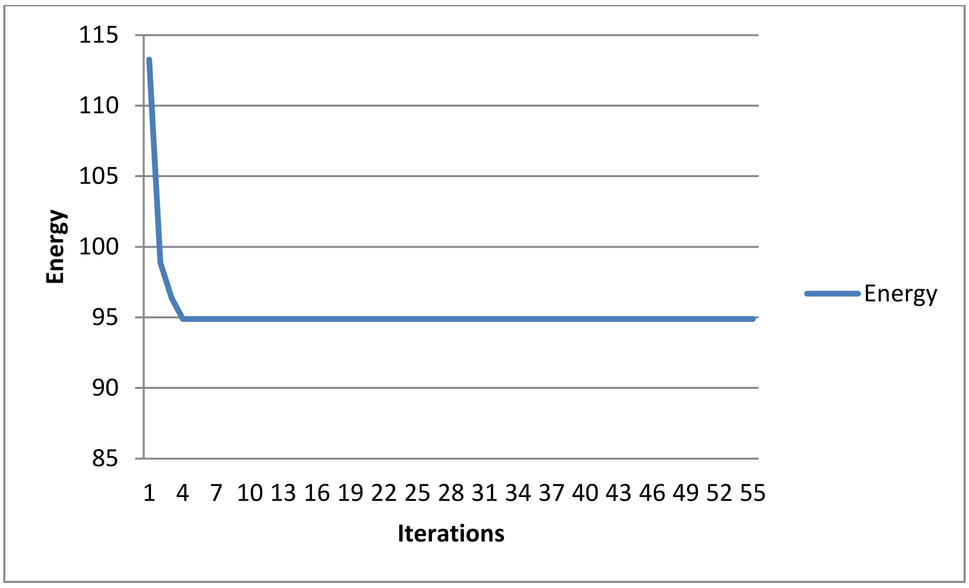


Figure 5.6- Objective function value Greedy Algorithm K_{max}= 18

Comparing the results obtained by simultaneous-operations SA, Incremental SA algorithms show dramatic performance gain. The graphs show uphill movements followed by farther drop in the energy function values. These uphill moves allow the solution to jump out of the local minima and head towards a global optimum configuration. Whereas the results obtained by the greedy algorithm get stuck in a local minima this is because the algorithm allows strict downhill moves.

Further examination of the results show, that simultaneous-operations SA Algorithm converges to lower energy values compared to the values obtained by the Incremental SA algorithm. And this is valid throughout the entire set of experiments with different test case length. Table 5-1 shows the energy values for different test case lengths while generating a test suite of 50 test cases.

Table 5.1 Energy Values for the three algorithms for different K values for WebApp-1.

Max number of events in Test cases	**Simultaneous-operation SA**	**Incremental SA**	**Greedy Algorithm**
k= 8	0.7505	0.8423	27.1832
k=9	0.7873	1.0421	30.2149
k=10	0.8596	1.1156	40.7051
k=18	1.2922	1.5328	94.8738
k=20	1.4265	1.6623	113.3677

.

Influenced by the previous observation we plan a new set of experiments to study test suite composition and cost effectiveness on a bigger FSM having 50 States and 270 semantically interacting events leading transitions between states.[Appendix IV]

Greedy, incremental simulated annealing and simultaneous-operation simulated annealing algorithms are run, to generate an optimized test suite of 60 test cases, each test case having a maximum of 20 events.

As to our intuition the simultaneous-operation simulated annealing successfully optimizes the test suite configuration and performs better than the incremental one. The energy function of the simultaneous-operation converges to a lower energy value then the incremental simulated annealing, generating an optimized test suite that insures diversity of test cases and continuous sequence of events. The greedy algorithms failed to generate continuous sequence of events of the maximum length of the test case. [Appendix VI] shows the obtained test case of simultaneous-operation simulated annealing

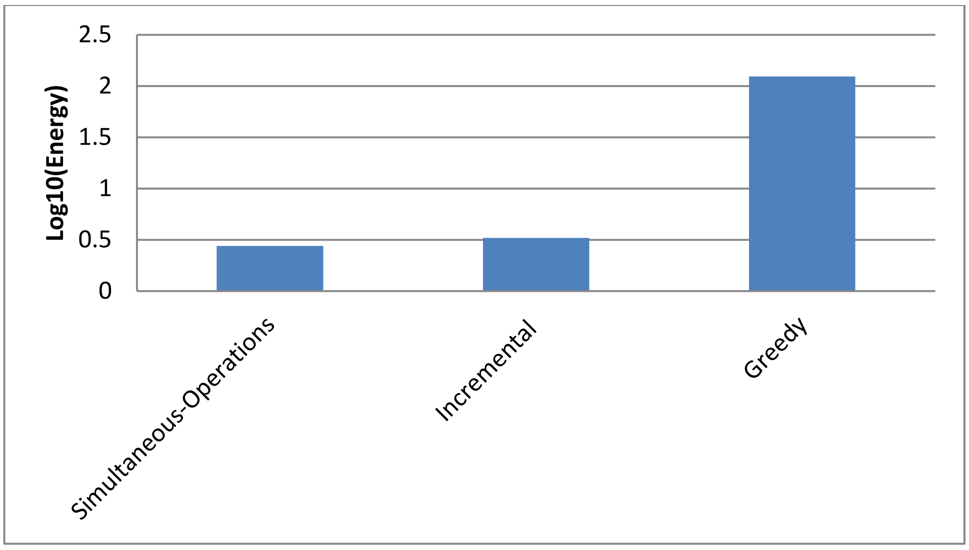


Figure 5.7 -Energy values for test suites of size 60 with a test case length k=20 using Simultaneous- Operations SA, Incremental SA and Greedy algorithms for WebApp-2 .

CHAPTER SIX

Conclusion

We have presented a testing technique that addresses the complexity of Web 2.0 applications. We have also modeled the dynamic features of Web 2.0 using state transition diagrams. Our model represents the important feature of the application as weights that are assigned to events. Test cases are generated as sequences of semantically interacting events using a simulated annealing algorithm. We also formulated an objective function that is based on the capability of test cases to provide high coverage of events, high diversity of events covered, and definite continuity of events. The experimental results show that the proposed simulated annealing algorithms generate more effective test cases than a previous hill-climbing algorithm. However, simultaneous-operation simulated annealing algorithm gives better results than the incremental simulated annealing.

The fact that simultaneous-operation simulated annealing has an edge over the incremental algorithm is not unexpected. Since the incremental algorithm will be bound to sequences generated in the preceding iterations and that previous decision will be penalizing newer configurations. Whereas in the simultaneous operations any decision is taken is not permanent and is subject to change during the remaining cycle of iterations thus making the decision taking more flexible.

The proposed technique proves its capability to handle different graph sizes and the strategy in inferring Finite State Machine out of the WEB 2.0 web application minimizes the manual interference need to perform state based testing.

More importantly the set of the test cases generated by the simultaneous-operation simulated annealing, which is associated with an optimal combination of coverage and diversity values, provides us with confidence in the effectiveness of these tests. This is a significant improvement over the previous work done on hill-climbing algorithm which results in a local optimum solution.

References

[1] A.Marchetto, P. Tonella, and F. Ricca. “State-based testing of ajax web applications,” in *Proc. of IEEE International Conference on Software Testing (ICST)*, Lillehammer, Norway, April 2008, pp. 121-130.

[2] A.Marchetto, P. Tonella. “Search-based testing of ajax web applications,” in *Proc. of IEEE Search Based Software Engineering, 2009 1st International Symposium*, 2009, pp. 3-12.

[3] T. O'Reilly, (2005, September 30). Design Patterns and Business Models for the Next Generation of Software [Online]. Available: http://oreilly.com/web2/archive/what-is-web-20.html

[4] P. Hegaret, (2005, January 19). Document Object Model (DOM). [Online]. Available: *http://www.w3.org/DOM*

[5] A.Andrews, J.Offutt, and T.Alexander. “Testing web applications by modeling with FSMs,” *Software and System Modeling*, vol.4 (3), 2005, pp. 326-345.

[6] A.Tarhini, N. Mansour, and H. Fouchal.“Regression testing web services based applications*,” IEEE International Conference Computer Systems and Applications 2006*, 2006, pp. 163-170.

[7] A.Tarhini, N. Mansour, and H. Fouchal. “Testing and regression testing for web services based applications,” *International Journal of Computing & Information Technology (IJCIT)*, vol.2 (2), pp. 195 – 217, 2010.

[8] G. Di Lucca, A. Fasolino, A. F. Faralli, and U. Carlini. “Testing web applications.”In *Proc. of the International Conference on Software Maintenance (ICSM)*, 2002, pp. 310-319.

[9] S. Elbaum, G. Rothermel, S. Karre, and M. Fisher. “Leveraging user session data to support web application testing,” *IEEE Transactions on Software Engineering*, 2005, vol.31 (3), pp. 187-202.

[10] Business Internet Group of San Francisco, (2003, Febreury 9), The BIG-SF Report on Government Web Application Integrity. [Online]Available: http://www.tealeaf.com/downloads/news/analyst_report/BIG-SF_Report_Gov_2003-05.pdf

[11] B. Fejes, (2004, April 19). Test Web Applications with HttpUnit. Available: http://www.javaworld.com/javaworld/jw-04-2004/jw-0419-httpunit.html.

[12] S. Kirkpatrick, C. Gelatt, and M.Vecchi. "Optimization by simulated Annealing", *Science*, vol. 220 (4598), pp. 671-680, 2006.

[13] B. Nikolik, "Test Diversity" *Information and Software Technology*, vol. 48 (3), pp. 1083–1094, 2006.

[14] A. Kolawa, D.Huizinga, *Automated Defect Prevention: Best Practices in Software Management*, New Jersey :Wiley-IEEE Computer Society Press, 2007, pp. 254.

[15] R. Ferguson, and B. Korel,(2006), "The chaining approach for software test data generation." *ACM Trans. Software Engineering Methodol*, vol. 5 (1), pp.63-86, 2006.

[16] S.Petrenko, S. Boroday, and R. Groz. "Confirming configurations in EFSM testing," *IEEE Trans. Software Engineering*, vol. 30, pp. 29-42, 2004.

[17] M.Memon, M. Pollack, and L. Soffa. "Hierarchical GUI test case generation using automated planning," *IEEE Trans. Software Engineering*, vol. 27 (2), pp. 144–155, 2001.

APPENDIX

APPENDIX I Iteration traces for generating a test suite of 50 test cases with maximum test case length K=18 using Simultaneous Operations Simulated annealing using the FSM of WebApp-1.

Iteration	Discontinuity	Lack of Diversity	Lack of Coverage	Energy
1	63.5	2.520488131	0.475	66.4954881
2	33.5	4.229256486	0.48125	38.2105065
3	15	4.859821028	0.475	20.334821
4	8	5.542144027	0.487666667	14.0298107
5	4	5.861451221	0.475	10.3364512
6	2	6.327152631	0.494256757	8.82140939
7	2	6.825896309	0.515140845	9.34103715
8	2	7.108998553	0.501027397	9.61002595
9	1.5	7.571516388	0.507986111	9.5795025
10	0.5	7.948686081	0.5225	8.97118608
11	0.5	8.546950358	0.501027397	9.54797776
12	0	8.448985763	0.501027397	8.95001316
13	0	8.37364081	0.537867647	8.91150846
14	0	8.094850858	0.501027397	8.59587826
15	0	8.16121072	0.494256757	8.65546748
16	0	8.156537894	0.537867647	8.69440554
17	0	8.024048879	0.545895522	8.5699444
18	0	7.560860429	0.494256757	8.05511719
19	0	7.173761943	0.48125	7.65501194
20	0	7.440622314	0.494256757	7.93487907
21	0	7.618422436	0.494256757	8.11267919
22	0	7.513179115	0.494256757	8.00743587
23	0	7.318836002	0.475	7.793836
24	0	7.734039075	0.501027397	8.23506647
25	0	7.593771159	0.501027397	8.09479856
26	0	7.51592379	0.507986111	8.0239099
27	0	7.606353293	0.515140845	8.12149414
28	0	7.678646392	0.507986111	8.1866325
29	0	7.37031956	0.530072464	7.90039202
30	0	7.242089507	0.515140845	7.75723035
31	0	6.962155587	0.487666667	7.44982225

32	0	6.679285921	0.48125	7.16053592
33	0	6.411151255	0.48125	6.89240126
34	0	6.240321339	0.487666667	6.72798801
35	0	6.167990793	0.48125	6.64924079
36	0	6.013244583	0.48125	6.49449458
37	0	5.586847091	0.475	6.06184709
38	0	5.580578861	0.487666667	6.06824553
39	0	5.152825479	0.48125	5.63407548
40	0	4.812910805	0.487666667	5.30057747
41	0	5.136059815	0.475	5.61105981
42	0	4.768947517	0.475	5.24394752
43	0	4.694716224	0.48125	5.17596622
44	0	4.303093122	0.475	4.77809312
45	0	3.87593349	0.487666667	4.36360016
46	0	3.871092923	0.475	4.34609292
47	0	3.773308948	0.475	4.24830895
48	0	3.536998787	0.48125	4.01824879
49	0	3.256203375	0.475	3.73120338
50	0	2.995890589	0.475	3.47089059
51	0	3.000268391	0.475	3.47526839
52	0	2.754153304	0.475	3.2291533
53	0	2.668868753	0.48125	3.15011875
54	0	2.578102097	0.475	3.0531021
55	0	2.484272614	0.48125	2.96552261
56	0	2.444199341	0.48125	2.92544934
57	0	2.367564238	0.475	2.84256424
58	0	2.177810924	0.475	2.65281092
59	0	2.194677749	0.475	2.66967775
60	0	2.214804827	0.475	2.68980483
61	0	2.017823189	0.475	2.49282319
62	0	1.907972856	0.475	2.38297286
63	0	1.951694244	0.475	2.42669424
64	0	1.766454194	0.48125	2.24770419
65	0	1.874596069	0.475	2.34959607
66	0	1.798363261	0.475	2.27336326
67	0	1.686078415	0.475	2.16107841
68	0	1.535125539	0.475	2.01012554
69	0	1.552130929	0.475	2.02713093
70	0	1.484624673	0.475	1.95962467
71	0	1.438874706	0.475	1.91387471
72	0	1.418753122	0.475	1.89375312
73	0	1.413899013	0.475	1.88889901
74	0	1.42007409	0.475	1.89507409
75	0	1.416989915	0.475	1.89198992
76	0	1.39655305	0.475	1.87155305

77	0	1.396105447	0.475	1.87110545
78	0	1.317805153	0.475	1.79280515
79	0	1.32772754	0.475	1.80272754
80	0	1.300138616	0.475	1.77513862
81	0	1.259110964	0.475	1.73411096
82	0	1.246639651	0.475	1.72163965
83	0	1.250644002	0.475	1.725644
84	0	1.192837131	0.475	1.66783713
85	0	1.260103337	0.475	1.73510334
86	0	1.274405909	0.475	1.74940591
87	0	1.197021479	0.475	1.67202148
88	0	1.17008992	0.475	1.64508992
89	0	1.183368252	0.475	1.65836825
90	0	1.167951378	0.475	1.64295138
91	0	1.143617252	0.475	1.61861725
92	0	1.166344898	0.475	1.6413449
93	0	1.170623945	0.475	1.64562394
94	0	1.096179921	0.475	1.57117992
95	0	1.136490396	0.475	1.6114904
96	0	1.085292781	0.475	1.56029278
97	0	1.124882403	0.475	1.5998824
98	0	1.064946205	0.475	1.53994621
99	0	1.114836499	0.475	1.5898365
100	0	1.066119327	0.475	1.54111933
101	0	1.08586851	0.475	1.56086851
102	0	1.055514292	0.475	1.53051429
103	0	1.023650536	0.475	1.49865054
104	0	1.048384672	0.475	1.52338467
105	0	1.013834513	0.475	1.48883451
106	0	1.014450797	0.475	1.4894508
107	0	1.005788457	0.475	1.48078846
108	0	0.981891247	0.475	1.45689125
109	0	0.988236014	0.475	1.46323601
110	0	0.979979806	0.475	1.45497981
111	0	0.96778635	0.475	1.44278635
112	0	0.979979806	0.475	1.45497981
113	0	0.96778635	0.475	1.44278635
114	0	0.939606524	0.475	1.41460652
115	0	0.942263456	0.475	1.41726346
116	0	0.924180946	0.475	1.39918095
117	0	0.915347158	0.475	1.39034716
118	0	0.929575398	0.475	1.4045754
119	0	0.908493489	0.475	1.38349349
120	0	0.911241143	0.475	1.38624114
121	0	0.921471877	0.475	1.39647188

122	0	0.921471877	0.475	1.39647188
123	0	0.910555007	0.475	1.38555501
124	0	0.918754821	0.475	1.39375482
125	0	0.900894234	0.475	1.37589423
126	0	0.900894234	0.475	1.37589423
127	0	0.893929762	0.475	1.36892976
128	0	0.878413582	0.475	1.35341358
129	0	0.869833559	0.475	1.34483356
130	0	0.863342586	0.475	1.33834259
131	0	0.867675297	0.475	1.3426753
132	0	0.859715313	0.475	1.33471531
133	0	0.858260112	0.475	1.33326011
134	0	0.84282882	0.475	1.31782882
135	0	0.856802439	0.475	1.33180244
136	0	0.851680938	0.475	1.32668094
137	0	0.851680938	0.475	1.32668094
138	0	0.84874049	0.475	1.32374049
139	0	0.853879629	0.475	1.32887963
140	0	0.84578982	0.475	1.32078982
141	0	0.851680938	0.475	1.32668094
142	0	0.843570045	0.475	1.31857004
143	0	0.84578982	0.475	1.32078982
144	0	0.842086943	0.475	1.31708694
145	0	0.84282882	0.475	1.31782882
146	0	0.839112877	0.475	1.31411288
147	0	0.839112877	0.475	1.31411288
148	0	0.833882738	0.475	1.30888274
149	0	0.83163118	0.475	1.30663118
150	0	0.834631907	0.475	1.30963191
151	0	0.840601225	0.475	1.31560123
152	0	0.833882738	0.475	1.30888274
153	0	0.833132895	0.475	1.30813289
154	0	0.832382376	0.475	1.30738238
155	0	0.827864977	0.475	1.30286498
156	0	0.824081562	0.475	1.29908156
157	0	0.822563323	0.475	1.29756332
158	0	0.821042277	0.475	1.29604228
159	0	0.822563323	0.475	1.29756332
160	0	0.818755409	0.475	1.29375541
161	0	0.818755409	0.475	1.29375541
162	0	0.821042277	0.475	1.29604228
163	0	0.821042277	0.475	1.29604228
164	0	0.817991699	0.475	1.2929917
165	0	0.819518407	0.475	1.29451841
166	0	0.821803152	0.475	1.29680315

167	0	0.823322792	0.475	1.29832279
168	0	0.817991699	0.475	1.2929917
169	0	0.819518407	0.475	1.29451841
170	0	0.821803152	0.475	1.29680315
171	0	0.820280696	0.475	1.2952807
172	0	0.820280696	0.475	1.2952807
173	0	0.818755409	0.475	1.29375541
174	0	0.818755409	0.475	1.29375541
175	0	0.818755409	0.475	1.29375541
176	0	0.818755409	0.475	1.29375541
177	0	0.817991699	0.475	1.2929917
178	0	0.817227276	0.475	1.29222728
179	0	0.819518407	0.475	1.29451841
180	0	0.817991699	0.475	1.2929917
181	0	0.818755409	0.475	1.29375541
182	0	0.818755409	0.475	1.29375541
183	0	0.818755409	0.475	1.29375541
184	0	0.818755409	0.475	1.29375541
185	0	0.818755409	0.475	1.29375541
186	0	0.818755409	0.475	1.29375541
187	0	0.817991699	0.475	1.2929917
188	0	0.817991699	0.475	1.2929917
189	0	0.818755409	0.475	1.29375541
190	0	0.817991699	0.475	1.2929917
191	0	0.817991699	0.475	1.2929917
192	0	0.817991699	0.475	1.2929917
193	0	0.817991699	0.475	1.2929917
194	0	0.817227276	0.475	1.29222728
195	0	0.817227276	0.475	1.29222728
196	0	0.817991699	0.475	1.2929917
197	0	0.817227276	0.475	1.29222728
198	0	0.817227276	0.475	1.29222728
199	0	0.817227276	0.475	1.29222728
200	0	0.817227276	0.475	1.29222728
201	0	0.817227276	0.475	1.29222728
202	0	0.817227276	0.475	1.29222728
203	0	0.817227276	0.475	1.29222728
204	0	0.817227276	0.475	1.29222728
205	0	0.817227276	0.475	1.29222728

APPENDIX II The test suite of 50 test cases with maximum test case length K=18 obtained using Simultaneous Operations Simulated annealing, using the small graph of Web Application 1.

Test case#	Event Sequences
1	55--53--55--53--58--61--51--53--57--59--2--15--17--20--17--21--24--25
2	50--48--43--46--49--41--3--52--58--61--52--56--83--81--83--82--84--84
3	84--19--24--25--16--18--20--16--18--22--25--17--21--24--26--29--32--84
4	45--41--1--7--11--14--10--9--14--11--13--4--64--67--69--71--83--83
5	48--43--46--50--47--42--43--46--49--41--2--15--17--21--24--26--30--35
6	19--24--27--41--1--7--11--13--1--6--5--9--13--4--63--0--5--84
7	43--45--41--3--51--53--57--59--2--15--17--22--25--17--20--16--19--23
8	82--83--81--84--72--66--75--68--73--74--64--68--72--63--1--6--5--84
9	82--19--24--26--31--36--28--26--30--34--39--36--31--37--33--32--40--29
10	48--44--50--47--42--43--45--42--43--46--49--42--43--45--42--43--46--49
11	81--84--58--61--51--54--62--59--4--63--4--63--2--15--16--18--21--23
12	20--16--18--22--26--30--34--38--30--35--40--29--32--40--31--36--29--32
13	18--22--27--41--3--52--55--53--57--59--2--15--16--19--24--26--29--32
14	33--32--40--30--35--40--28--27--41--4--64--68--73--74--63--0--5--9
15	76--69--70--68--73--74--65--70--68--73--74--65--71--69--71--69--71--84
16	60--55--54--61--51--54--61--52--57--59--1--7--11--13--4--66--75--67
17	37--34--39--36--31--37--34--39--37--35--40--31--36--30--34--39--37--33
18	72--65--71--69--70--68--72--66--75--68--72--66--75--68--72--65--71--82
19	58--62--60--57--59--4--63--3--52--55--54--61--51--54--62--60--56--83
20	38--30--34--38--31--36--28--27--41--3--52--55--54--61--51--53--56--82
21	8--12--13--1--7--11--13--0--5--9--13--1--6--5--9--14--10--84
22	20--16--18--22--26--28--27--42--44--49--42--44--50--48--43--46--50--47
23	83--84--82--58--62--60--58--62--59--3--51--54--61--51--54--62--60--56
24	81--84--81--8--12--13--0--5--9--13--4--63--1--8--12--14--10--81
25	81--84--82--81--55--54--62--59--3--52--55--53--58--62--59--1--6--5
26	48--44--50--47--42--44--49--41--2--15--16--19--23--15--17--21--23--82
27	39--37--34--38--31--36--30--35--40--28--25--17--21--24--25--17--22--27
28	45--42--43--45--42--43--46--50--48--44--49--41--1--6--5--9--14--10
29	73--74--64--68--72--66--75--68--72--63--3--51--53--58--61--51--53--56
30	73--75--67--69--70--68--72--66--74--65--71--69--70--67--69--70--67--82
31	33--32--40--30--35--40--28--25--17--21--24--25--16--19--23--15--17--20
32	37--34--38--30--33--32--40--31--36--28--25--17--20--16--18--21--23--81
33	52--55--54--61--52--57--59--4--66--75--67--69--70--67--83--84--84--83
34	34--39--37--35--40--28--27--41--0--5--9--13--1--8--12--14--11--14
35	57--60--57--60--57--60--57--60--55--53--57--59--4--66--76--81--84--82
36	48--44--50--47--41--4--65--70--68--72--66--74--66--74--63--1--7--10
37	8--12--14--11--13--1--7--11--14--11--13--3--52--58--61--51--53--56

38	81--82--8--12--14--10--9--13--4--66--74--63--3--52--58--61--52--56
39	84--19--23--15--16--18--20--16--18--22--26--28--25--16--18--21--23--84
40	81--18--20--17--21--24--27--41--4--64--68--73--76--69--70--67--83--83
41	46--50--47--42--44--50--47--41--3--51--54--62--59--2--15--16--19--23
42	76--69--71--69--70--68--72--66--74--63--1--8--12--13--0--5--83--83
43	48--43--46--49--41--1--7--10--9--13--4--63--2--15--17--22--26--29
44	8--12--14--11--13--0--5--9--13--3--52--55--53--58--62--60--56--84
45	18--21--24--26--30--35--40--28--26--31--37--34--39--36--31--36--30--33
46	45--42--44--50--48--44--49--42--44--50--47--41--4--65--70--67--69--71
47	48--43--45--41--4--64--67--69--70--68--73--74--66--74--65--71--82--81
48	2--15--17--22--26--28--26--28--27--41--4--64--68--73--75--68--73--76
49	55--54--62--60--57--60--58--62--59--1--8--12--14--10--9--14--10--84
50	37--34--38--31--36--28--26--28--26--29--32--40--31--36--31--37--33--32

APPENDIX III Iteration traces for generating a test suite of 60 test cases with maximum test case length K=20 using Simultaneous Operations Simulated annealing using the FSM of WebApp-2.

Iteration	Discontinuity	Lack of Diversity	Lack of Coverage	Energy
1	130.5	2.351254874	0.468925234	133.3201801
2	84	2.888494328	0.473349057	87.36184338
3	66.5	3.065680917	0.494334975	70.06001589
4	52	3.576786195	0.498014888	56.07480108
5	42.5	3.669250534	0.487135922	46.65638646
6	35.5	3.803077633	0.503007519	39.80608515
7	29	3.915788488	0.510687023	33.42647551
8	26.5	3.820785192	0.504271357	30.82505655
9	24	3.690988957	0.503007519	28.19399648
10	22.5	3.514740315	0.500498753	26.51523907
11	20	3.491188835	0.491911765	23.9831006
12	19.5	3.753451676	0.514615385	23.76806706
13	17.5	3.516162608	0.506818182	21.52298079
14	16.5	3.514740315	0.509390863	20.52413118
15	15.5	3.386502544	0.504271357	19.3907739
16	14	3.387978672	0.495555556	17.88353423
17	13	3.341167383	0.504271357	16.84543874
18	13	3.213004744	0.503007519	16.71601226
19	12.5	3.272369093	0.503007519	16.27537661
20	12	3.274660209	0.510687023	15.78534723
21	10.5	3.385764239	0.511989796	14.39775404
22	9.5	3.205214421	0.500498753	13.20571317
23	9	3.185655267	0.510687023	12.69634229
24	8.5	3.21844675	0.503007519	12.22145427
25	8	3.238579856	0.50175	11.74032986
26	7	3.259355685	0.514615385	10.77397107
27	7	3.116632715	0.499253731	10.61588645
28	6	3.059150124	0.511989796	9.57113992
29	6	3.098128384	0.508101266	9.60622965
30	6	3.220776224	0.518604651	9.739380875
31	5.5	3.044404619	0.526771654	9.071176272
32	5.5	3.095706621	0.509390863	9.105097484
33	5.5	3.063233501	0.518604651	9.081838152
34	5	3.031237286	0.521298701	8.552535987
35	5	3.193493304	0.529551451	8.723044755
36	5	2.982180324	0.515938303	8.498118627
37	4.5	3.080324574	0.52539267	8.105717244

38	4.5	3.013038248	0.513299233	8.026337481
39	4.5	3.086000564	0.515938303	8.101938868
40	4.5	3.104577183	0.515938303	8.120515486
41	3.5	2.945063579	0.519948187	6.965011765
42	3	2.903170591	0.514615385	6.417785976
43	3	2.978825185	0.510687023	6.489512208
44	2.5	2.93315521	0.508101266	5.941256475
45	2.5	2.930597121	0.514615385	5.945212505
46	2.5	2.968737018	0.511989796	5.980726814
47	2.5	2.928890487	0.509390863	5.93828135
48	2.5	2.922909421	0.521298701	5.944208122
49	2.5	2.920342357	0.521298701	5.941641058
50	2.5	2.903170591	0.518604651	5.921775243
51	2.5	2.820177208	0.509390863	5.829568071
52	2	2.7736978	0.510687023	5.284384822
53	2	2.8501578	0.514615385	5.364773184
54	2	2.88503024	0.513299233	5.398329472
55	2	2.845768698	0.509390863	5.355159561
56	2	2.837851209	0.511989796	5.349841005
57	2	2.764669869	0.517268041	5.28193791
58	2	2.80951944	0.513299233	5.322818673
59	1.5	2.74925435	0.508101266	4.757355616
60	1.5	2.750163538	0.508101266	4.758264803
61	1.5	2.774598977	0.509390863	4.78398984
62	1.5	2.706178021	0.494334975	4.700512997
63	1.5	2.67271388	0.503007519	4.675721399
64	1	2.666158188	0.498014888	4.164173076
65	1	2.703405164	0.498014888	4.201420053
66	1	2.67271388	0.496782178	4.169496058
67	1	2.653940369	0.494334975	4.148275345
68	1	2.663343666	0.495555556	4.158899222
69	1	2.649226205	0.499253731	4.148479936
70	1	2.630285057	0.496782178	4.127067236
71	1	2.592951886	0.491911765	4.084863651
72	0.5	2.621716896	0.499253731	3.620970627
73	0.5	2.621716896	0.494334975	3.616051871
74	0.5	2.621716896	0.498014888	3.619731784
75	0.5	2.617899823	0.503007519	3.620907342
76	0.5	2.624576058	0.500498753	3.625074811
77	0.5	2.652055709	0.496782178	3.648837887
78	0.5	2.591987555	0.495555556	3.58754311
79	0.5	2.583292373	0.496782178	3.580074551
80	0.5	2.581356133	0.494334975	3.575691108
81	0.5	2.589092405	0.494334975	3.583427381
82	0.5	2.572625018	0.494334975	3.566959993

83	0.5	2.586194015	0.495555556	3.581749571
84	0.5	2.56581361	0.493120393	3.558934003
85	0.5	2.566787775	0.496782178	3.563569953
86	0.5	2.551156499	0.500498753	3.551655252
87	0.5	2.515631031	0.495555556	3.511186586
88	0.5	2.507668137	0.498014888	3.505683025
89	0	2.506670996	0.496782178	3.003453174
90	0	2.505673459	0.494334975	3.000008434
91	0	2.513642672	0.493120393	3.006763065
92	0	2.502678462	0.498014888	3.00069335
93	0	2.470505916	0.494334975	2.964840892
94	0	2.481612275	0.493120393	2.974732668
95	0	2.460365721	0.491911765	2.952277486
96	0	2.456297922	0.490709046	2.947006969
97	0	2.444053903	0.493120393	2.937174297
98	0	2.429691232	0.489512195	2.919203427
99	0	2.419379979	0.488321168	2.907701147
100	0	2.420413081	0.485956416	2.906369498
101	0	2.395495665	0.485956416	2.881452082
102	0	2.402789937	0.485956416	2.888746354
103	0	2.428662077	0.488321168	2.916983245
104	0	2.395495665	0.484782609	2.880278274
105	0	2.385036579	0.485956416	2.870992995
106	0	2.378739053	0.487135922	2.865874976
107	0	2.386084551	0.484782609	2.870867159
108	0	2.370316326	0.487135922	2.857452248
109	0	2.355504083	0.487135922	2.842640005
110	0	2.344866624	0.484782609	2.829649232
111	0	2.353380437	0.487135922	2.84051636
112	0	2.357625815	0.487135922	2.844761737
113	0	2.363979586	0.488321168	2.852300754
114	0	2.353380437	0.488321168	2.841701605
115	0	2.361863562	0.488321168	2.850184729
116	0	2.348062921	0.487135922	2.835198844
117	0	2.358685965	0.484782609	2.843468574
118	0	2.353380437	0.484782609	2.838163046
119	0	2.337391598	0.485956416	2.823348015
120	0	2.325596587	0.484782609	2.810379195
121	0	2.328819332	0.487135922	2.815955255
122	0	2.328819332	0.484782609	2.813601941
123	0	2.321292632	0.484782609	2.806075241
124	0	2.320215396	0.484782609	2.804998005
125	0	2.318059422	0.484782609	2.802842031
126	0	2.316980683	0.484782609	2.801763291
127	0	2.321292632	0.482451923	2.803744555

128	0	2.319137659	0.482451923	2.801589582
129	0	2.298564657	0.481294964	2.779859621
130	0	2.300738899	0.481294964	2.782033863
131	0	2.293120032	0.481294964	2.774414996
132	0	2.288755007	0.481294964	2.770049971
133	0	2.296388356	0.482451923	2.778840279
134	0	2.293120032	0.482451923	2.775571956
135	0	2.288755007	0.481294964	2.770049971
136	0	2.292029555	0.481294964	2.773324519
137	0	2.285475767	0.481294964	2.766770731
138	0	2.284381641	0.481294964	2.765676605
139	0	2.285475767	0.481294964	2.766770731
140	0	2.283286991	0.481294964	2.764581955
141	0	2.281096114	0.482451923	2.763548037
142	0	2.279999887	0.481294964	2.761294851
143	0	2.276708036	0.481294964	2.758003
144	0	2.281096114	0.481294964	2.762391078
145	0	2.281096114	0.481294964	2.762391078
146	0	2.281096114	0.481294964	2.762391078
147	0	2.275609695	0.481294964	2.756904659
148	0	2.276708036	0.481294964	2.758003
149	0	2.276708036	0.481294964	2.758003
150	0	2.276708036	0.481294964	2.758003
151	0	2.274510823	0.481294964	2.755805787
152	0	2.275609695	0.481294964	2.756904659
153	0	2.275609695	0.481294964	2.756904659
154	0	2.275609695	0.481294964	2.756904659
155	0	2.275609695	0.481294964	2.756904659
156	0	2.276708036	0.481294964	2.758003
157	0	2.277805848	0.481294964	2.759100812
158	0	2.278903132	0.481294964	2.760198096
159	0	2.277805848	0.481294964	2.759100812
160	0	2.274510823	0.481294964	2.755805787
161	0	2.274510823	0.481294964	2.755805787
162	0	2.274510823	0.481294964	2.755805787
163	0	2.274510823	0.481294964	2.755805787
164	0	2.27341142	0.481294964	2.754706384
165	0	2.27341142	0.481294964	2.754706384
166	0	2.272311485	0.481294964	2.753606449
167	0	2.272311485	0.481294964	2.753606449
168	0	2.272311485	0.481294964	2.753606449
169	0	2.272311485	0.481294964	2.753606449
170	0	2.272311485	0.481294964	2.753606449
171	0	2.272311485	0.481294964	2.753606449
172	0	2.272311485	0.481294964	2.753606449

173	0	2.272311485	0.481294964	2.753606449
174	0	2.272311485	0.481294964	2.753606449
175	0	2.272311485	0.481294964	2.753606449
176	0	2.272311485	0.481294964	2.753606449
177	0	2.272311485	0.481294964	2.753606449
178	0	2.272311485	0.481294964	2.753606449
179	0	2.272311485	0.481294964	2.753606449
180	0	2.272311485	0.481294964	2.753606449
181	0	2.272311485	0.481294964	2.753606449
182	0	2.272311485	0.481294964	2.753606449
183	0	2.272311485	0.481294964	2.753606449
184	0	2.272311485	0.481294964	2.753606449
185	0	2.272311485	0.481294964	2.753606449
186	0	2.272311485	0.481294964	2.753606449
187	0	2.272311485	0.481294964	2.753606449
188	0	2.272311485	0.481294964	2.753606449
189	0	2.272311485	0.481294964	2.753606449

APPENDIX IV: The test suite of 60 test cases with maximum test case length K=20 obtained using Simultaneous Operations Simulated Annealing on WebApp-2

Test Case #	**Event Sequences**
1	243--220--132--137--141--13--244--228--125--169--99--150--236--35--81--69--138--172--187
2	221--209--233--243--220--133--145--149--129--47--69--137--146--172--195--204--79--15
3	267--238--40--35--80--47--67--104--96--103--157--241--196--225--77--85--1--18--269--266
4	269--266--232--217--81--69--137--144--115--125--167--68--134--176--3--7--16--23
5	207--133--141--11--141--13--243--215--49--52--46--54--58--65--99--148--84--116
6	12--218--101--73--77--86--107--161--156--160--138--169--94--74--99--149--132--135
7	269--179--122--115--125--172--190--54--58--67--104--93--35--80--47--69--138--164
8	132--137--141--12--214--34--43--17--35--80--47--65--93--33--24--7--16--23
9	267--183--220--134--182--208--192--103--154--107--163--181--196--224--74--92--29--48
10	161--156--162--173--156--161--155--151--119--115--125--167--65--96--102--104--98--128
11	268--242--209--232--218--103--157--241--192--103--157--239--50--71--27--18--1--18
12	227--99--148--84--119--115--126--181--192--102--104--94--72--41--44--46--55--72
13	266--269--269--266--267--224--70--1--19--20--0--3--6--14--23--0--3--6-265--264
14	268--62--57--47--62--59--77--88--114--82--1--19--22--30--0--3--7--15--265--269
15	227--97--115--122--115--125--170--141--12--220--131--87--110--107--163--183--217--79
16	240--54--58--69--138--167--68--134--182--208--193--115--121--112--134--182--202--48
17	268--269--269--267--267--228--125--169--94--74--98--133--143--71--28--40--34--42
18	12--221--208--192--103--156--160--138--171--161--157--244--233--244--231--196--231--187
19	162--173--157--244--233--242--204--81--64--83--87--112--131--86--107--159--13--234
20	268--232--221--207--131--87--112--131--88--114--84--117--35--80--46--53--1--18
21	218--103--156--159--13--242--207--130--83--88--114--84--118--82--1--19--22--30
22	170--145--149--132--138--171--162--174--169--97--115--123--150--242--208--194--181--186
23	266--159--13--241--196--229--150--243--215--50--74--98--131--87--110--106--87--108
24	125--171--163--179--123--149--133--144--115--123--150--240--55--74--98--134--179--120
25	227--96--103--157--243--221--208--191--81--61--32--0--3--6--14--24--6--14
26	230--152--150--241--195--204--81--65--96--102--104--91--25--36--1--19--21--26
27	159--12--220--129--47--62--58--69--138--172--196--231--194--180--174--170--143--70
28	266--268--268--232--221--209--225--77--88--115--121--109--104--93--34--44--46--53
29	203--54--58--63--76--55--74--90--2--4--10--22--31--37--19--21--28--38
30	267--266--233--242--209--228--121--112--132--137--141--11--146--169--89--1--19--20
31	162--174--170--146--172--195--205--103--156--160--138--172--193--115--124--152--149--12
32	121--109--104--98--132--138--172--195--209--232--220--133--145--150--242--199--7--16
33	269--226--83--88--114--84--118--84--119--115--126--182--209--233--244--233--243--212
34	230--152--150--244--228--121--112--131--88--115--124--151--118--83--86--106--86--105
35	170--146--166--49--52--47--69--138--172--195--205--103--154--107--163--177--40--33
36	106--87--111--113--0--2--4--12--216--53--0--3--7--17--35--80--45--266--266--269
37	169--95--77--87--110--107--159--13--244--232--216--54--58--68--134--181--189--49
38	268--228--126--180--174--167--66--103--156--159--13--241--190--54--58--61--33--25--36--26
39	229--149--134--179--123--150--244--229--150--241--196--227--99--150--241--188--34--42--2
40	180--173--157--235--2--4--11--145--150--244--231--194--181--196--231--195--206--113

41	191--81--68--133--146--171--160--137--141--13--241--195--208--189--50--74--92--26
42	162--173--155--152--148--84--117--35--81--63--76--54--57--46--55--74--95--75-265--249
43	136--107--163--182--208--194--182--209--228--123--148--83--88--113--0--2--4--9
44	126--179--126--180--173--157--243--221--208--195--205--101--71--28--41--43--17--32
45	179--126--180--174--171--160--136--107--163--183--217--81--63--77--86--106--86--105
46	151--117--32--0--2--4--11--146--167--69--137--146--167--66--103--157--237--37
47	269--183--217--81--66--102--104--98--134--180--174--168--76--54--57--47--66--100
48	267--266-171--161--155--152--149--134--178--47--62--59--76--55--74--96--101--72--41--42
49	227--97--115--125--167--65--99--149--134--181--194--183--219--114--84--117--35--78--269
50	194--183--221--207--131--87--110--107--163--181--196--228--124--151--118--83--86--105
51	269--265--2--4--11--143--73--76--54--58--62--59--76--55--71--28--39--31--37--18--269--269
52	230--152--149--131--86--107--158--0--2--5--14--24--6--92--29--50--74--96--100--18
53	11--141--11--141--11--145--149--131--86--107--159--13--243--221--207--134--183--218--101
54	267--232--220--129--47--66--103--156--163--183--220--134--179--122--115--125--167--61--3
55	267--171--162--174--167--62--58--68--132--137--146--167--66--102--104--93--35--79--15--2
56	267--269--227--91--24--7--17--32--1--19--21--28--41--43--16--23--205--102--104--91
57	160--137--141--12--217--80--47--69--138--169--99--150--236--34--43--17--33--24--7--15
58	226--84--119--115--126--181--191--81--67--104--98--134--183--217--79--17--34--43--16--24
59	243--220--132--135--216--54--58--69--137--141--13--244--231--195--201--35--79--17--34--4
60	118--83--88--115--126--179--126--179--126--181--189--50--73--76--54--58--65--99--149--12