IntelligenTester –Test Sequence Optimization framework using Multi-Agents

D.Jeya Mala[1]
[1]Department of Computer Applications
Thiagarajar College of Engineering, Madurai, Tamil Nadu, India.
E-Mail : djmcse@tce.edu

Dr.V.Mohan [2]
[2] Prof.& Dean – Planning and Administration,
Thiagarajar College of Engineering, Madurai, Tamil Nadu, India
E-Mail : srubi77@yahoo.com

***Abstract* - Our paper focuses on the generation of optimal test sequences and test cases using Intelligent Agents for highly reliable systems. Test sequences support test case generation for these types of systems. Our system is modeled through UML state charts. Conventional test generation techniques do not worry about optimization and dynamic nature of such systems. In the case of highly reliable Software Testing, we can implement agents with sophisticated intellectual capabilities such as the ability to reason, learn, or plan. In our proposed approach, we developed agents namely Intelligent Search Agent (ISA) for optimal test sequence generation and Intelligent Test Case Optimization Agent (ITOA) based on HGA for optimal test case generation. Finally, we compared our results against existing algorithms. We registered our tool "IntelligenTester" under Java Research License (JRL) under the URL name https://intelligentester.dev.java.net.**

***Index Terms* – Intelligent Agents, Software Testing, UML (Unified Mark-up Language), HGA (Hybrid Genetic Algorithm)**

I. INTRODUCTION

The application of artificial intelligence (AI) techniques in software testing [6] to achieve Quality Software is an emerging area of research that brings about the cross fertilization of ideas across two domains such as AI and Software Engineering. A number of published works, for examples [2] and [3], have begun to examine the effective use of AI for SE related activities, which are inherently knowledge intensive and human-centered.

Hence, we applied Intelligent Agent based test sequence and test case generation and optimization. Results known from state-based conformance testing [5] and shortest paths in graph theory are used and extended to construct algorithms for test sequence generation and selection to cover the behavioral model of the software under test (SUT). The SUT and the sequence of states in which it is getting executed can be represented as a UML State Chart, which is used as a dynamic behavior model worldwide [7]. This State Chart is then converted to an Execution Sequence Graph (ESG).The graph is learned by the Agent and the agent searched the state space to find out only the optimal test sequences in the SUT [1][12]. Whenever a new state is added or removed from the SUT, the ISA will automatically identify it and will change the test sequence accordingly by getting the information from the environment, which is our SUT.

In HGA, we are combining both GA and Intelligent local search (GA-LS) approaches [29]. They are generally more effective than using stand alone versions of each method. Hybrid Genetic Algorithms is a population-based approach for heuristic search in optimization problems. They execute order of magnitude faster than traditional Genetic algorithms. The agents are developed using Java [14] in Eclipse IDE with JESS plug-in and JACK was used for modeling.

A. Intelligent Agents – An Introduction

Intelligent Agents are pieces of S/W that are designed to make computing and other tasks easier by assisting and acting on behalf of the user. The user can interact with the agent through the user interface and while the agent can sense and act according to the condition of the external environment. The agent performs its tasks by taking information from the environment in which it is working [13].

*"An **autonomous agent** is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future"[13][15]*

In the case of Software Testing [14], we can implement agents with sophisticated intellectual capabilities such as the ability to reason, learn, or plan. In addition, intelligent software agents can utilize extensive amounts of knowledge about their problem domain [15]. This means that the underlying agent architecture must support sophisticated reasoning, learning, planning, and knowledge representation [6].

In our approach, after setting up the UML state chart, the agent will convert it into the directed graph and the generation of optimal test sequences will be done automatically by searching through the state space. Then using HGA identifies the optimal test cases. The conversion, test sequence generation and test case generation algorithms are fully automatic and thus require no user supervision for its application, which makes it an intelligent tool.

II. Proposed Approach – Hybrid Optimization Framework

Our test optimization framework includes two important optimizations:

1. Test Sequence Optimization
2. Test Case Optimization

In the proposed framework, we are introducing an IntelligenTester framework, which will do the test sequence and test case optimization tasks. It consists of the following two agents:

- Intelligent Search Agent (ISA) – For test sequence optimization
- Intelligent Test Case Optimization Agent (ITOA) – For test cases optimization

Initially, the source code is given as input and from which the test sequences and test cases are generated. Among them only the optimal ones are identified and stored in the test database.

In our framework, we used the Coverage Analysis [4] for finding the coverage measure in the optimal test sequence identification process. To find the optimal test cases, we used Mutation score as a measure of identifying the best test cases. Finally, a test report will be generated and the result is also stored in the database.

Inside the IntelligenTester framework, the two agents are consisting of a set of processes which themselves are going to be done by separate software agents. The Intelligent Search Agent (ISA) is consisting of the following agents:

- SUT Reader Agent
- Converter Agent
- Feasible Test Sequence Generation Agent
- Optimal Test Sequence Generation Agent

Similar to that, the Intelligent Test Case Optimization Agent (ITOA) is consisting of the following agents:

- Test Case generator agent
- Test Case Evaluation Agent
- Test Case Selector agent
- Monitor and Report agent

III. Test Sequence Optimization - ISA

The test sequences are the set of execution paths in the SUT. Path testing involves the identification of all possible paths in the given system. Usually the number of paths in a SUT is infinitely many, and it is impossible to exercise all the paths in the system [8]. Hence, we need to find out the optimal number of test paths in the system so that the time and cost involved in the testing process will be reduced.

Here, we used an Intelligent Agent that will make the optimization of the paths by detecting both infeasible paths and redundant sub paths in the set of test sequences.

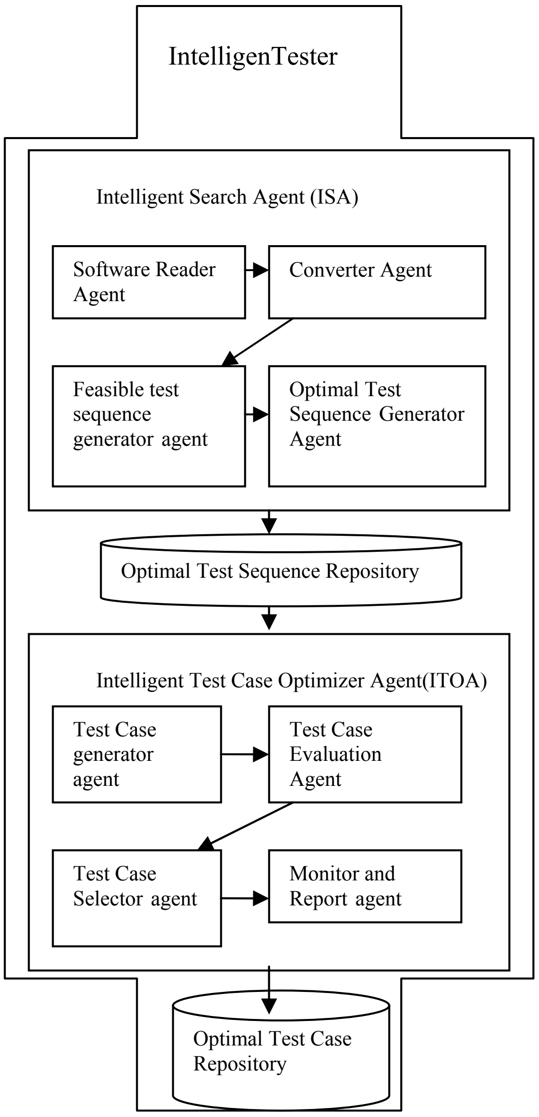

IntelligenTester
Intelligent Search Agent (ISA)
Software Reader Agent
Converter Agent
Feasible test sequence generator agent
Optimal Test Sequence Generator Agent
Optimal Test Sequence Repository
Intelligent Test Case Optimizer Agent(ITOA)
Test Case generator agent
Test Case Evaluation Agent
Test Case Selector agent
Monitor and Report agent
Optimal Test Case Repository

Fig. 1. Intelligent Test Optimization Framework

ISA will generate the list of test sequences from the set of world states and the transitions between them are provided as inputs. The set of world states and the interaction between them are stored in its knowledge base as perception sequences. Then the action of either to include a test sequence or remove it from the data base or the test sequence repository is done based on the repeated sub path identification.

A. Optimized Test Sequence Generation using ISA

Here the agent that we used is a Search Agent, which is a Java Swing application [14]. It takes the graph as input and produces the best-optimized paths depending upon the sum of the heuristic value on each node and the cost on each edge and at the same time covering all the nodes without the repetition of sub paths. Learning from both forward and backward searching and then lists the optimized paths until there are no more frontier nodes to consider does this.

The steps to be followed are:

1. Transformation of SUT into UML state chart notation;

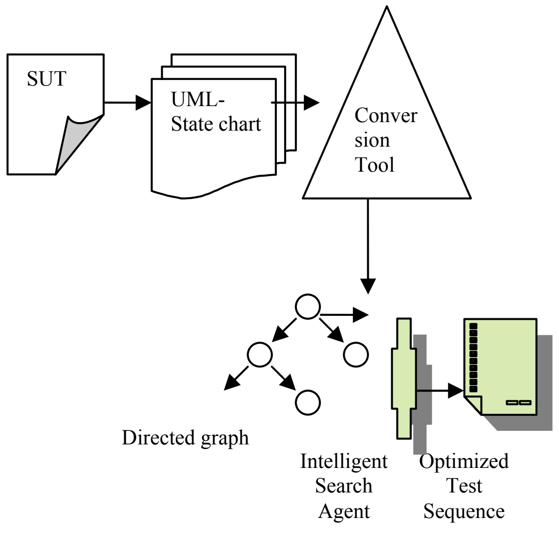


Fig.2. Optimized Test Sequence Generator –ISA architecture.

2. Converting the State chart into a directed graph and assigning weight to each of the edge connecting;
3. A mechanism for creating possible test sequences efficiently and a suitable criterion to stop solution generation;
4. A transition rule to insert or exclude the nodes in the execution sequence.
5. Generation of Optimized Test Sequences/ Paths

B. Internal Architecture of ISA

In the internal architecture as in Fig. 3, the ISA has a sensor, which is not an h/w sensor rather it is a software code that receives the nodes, edges, the cost associated with each edge and the node heuristics. The state module consists of all the states in the system.

From the perception history, the ISA will find the next state in the SUT and will find out the frontier nodes from that state. If there are more than one frontier nodes, the ISA will decide which frontier to choose now by checking what will be the consequences of taking the said action to reach those frontier nodes. Based on this, a happiness value is associated with each state. A state with higher happiness value is selected and then repetition of that state is checked from the existing test sequences.

These will be stored in the optimized test sequence repository and then this newly formulated test sequences will be provided as input to the ISA for the next step. This will be continued till we reach a system state of no more frontier nodes exist.

The criterion to be considered here is all the states are to be covered with minimum cost that is with less time. The proposed approach deals with the automatic generation of test sequences from the UML state chart by converting it into a directed graph form. Now, the generated test suite has to satisfy the following criteria:

1. Covering all states
2. Finding feasible path
3. Optimal test sequence generation

As per the Prometheus Methodology [27], we have developed the agent descriptor, the capability descriptor, percept descriptor, action descriptor and so on. The agent descriptor is shown below.

C. Reading of SUT – SUT Reader Agent

This agent will open up a dialog box for the user/tester to select the Software for which the optimal test sequences are to be generated.

D. Generating a directed graph – Converter Agent

State-based testing is a frequently used approach in software testing [9]. The main problems to be addressed are redundancy of test sequences and infeasible test sequences. First a directed graph for the given SUT is constructed. For which we had a tool to convert the state chart notation into a directed graph.

In order to assign weight to each of the edge, which is nothing but the processing / execution time or the likelihood function value to transit from one node to another node, we need to calculate it by System.currentTimeMillin () or by using the probability function to calculate the likelihood value. Since the intelligent agent software is written in Java, it is platform independent and can be used in any operating system.

E. Generation of all Feasible and Optimal test sequences generation – Intelligent Search Agent

The path length is the sum of the weights of the edges on that path rather than the number of edges. Each path could then be tested against the constraints and the first path satisfying all these constraints would be the optimal path. Then we will find the next optimal and so on till we ensure that all the nodes have been covered at least once.

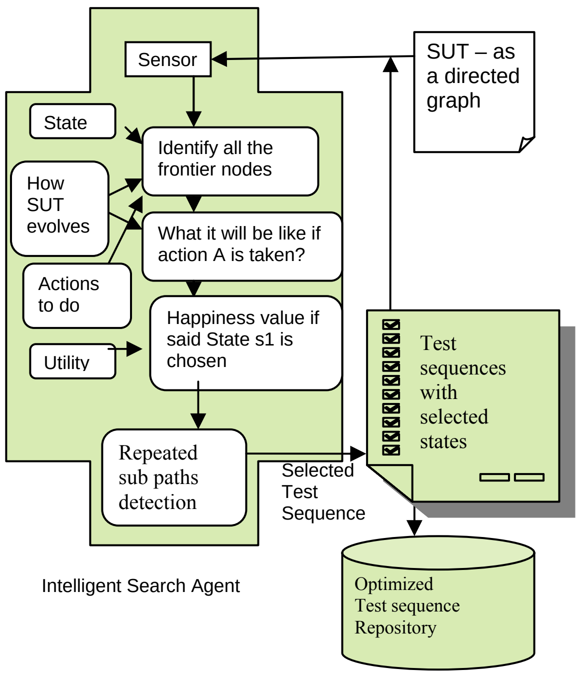

Sensor
SUT – as a directed graph
State
Identify all the frontier nodes
How SUT evolves
What it will be like if action A is taken?
Actions to do
Happiness value if said State s1 is chosen
Utility
Test sequences with selected states
Repeated sub paths detection
Selected Test Sequence
Intelligent Search Agent
Optimized Test sequence Repository

Fig 3. Internal Architecture of Intelligent Search Agent

IV. ISA Algorithm (Intelligent Search Agent Algorithm)

Usually, intelligent search algorithms [8] are applied in Path finding and Constraint Satisfaction. Path finding problems [5] are focused on finding paths from initial state to the goal state or the final state. Finding an efficient, optimal path between initial states to the final state is the goal to achieve. Constraint satisfaction [13] here is that, we should not repeat any sub paths, which have been already covered by the previous paths. In order to achieve effective searching algorithm [12], we need to specify the relative ness measure can be formulated by identifying the difference between happy and unhappy states. The said goal can be achieved using the following ways:

- Infeasible path detection
- Repeated sub paths detection
- Coverage of all states and branches at least once.

A. Infeasible Path Detection Algorithm

1. Convert the SUT into a UML State Chart notation.
2. Translate the State Chart into an Execution Sequence Graph (ESG). This graph is a directed graph in which each node is the state in the State Chart and edge between the nodes is the state transition between the states. Start from the initial node and to identify the next best node to explore, a feasibility checking will be done by calculating the edge weight w(e) and the likelihood/goodness value of each node.
3. The Edge Weight (w(e)) is calculated as follows:
   w(e) = Time taken to traverse from the current node to the next node
4. The Goodness criterion for each node is calculated as follows
   h(n) = probability / likelihood of choosing the said node
5. Happiness value (hv(n)) can be calculated as follows: **hv(n)= w(e) + h(n)**.
6. Based on this hv, the next state is selected among the set of world states.
7. While traversing to the next best node, the **hv(n)** is accumulated for that path, and if there is an alternate path from the previously rejected node to the next best node, then the total weight associated to that path will be calculated, as follows: hv(n1)+ hv(n2)+….hv(nm)
   If it seems to be the best one, then this path will be chosen as the optimal one.
8. If we reached the final node, then the first optimal path is selected.
9. Repeat steps 3 to 8 until we arrived at a state where we don’t have any more frontier states to explore.

B. Repeated Test Sequence Detection Algorithm

1. For each generated test sequence a similarity measure is calculated which is acting like a fitness value.
2. Fitness Function **f(x) = similarity (ti, tj),** where similarity (ti, tj) finds the distance between two paths ti and tj which extends the functionality of Hamming distance.
3. Given a target path and a current one, the similarity between them is calculated from n-order sets of ordered and even the cascaded branches for each paths being compared.
4. The symmetric difference between them is used to give the distance between them.
5. Then the distance is normalized to become a real number and the similarity between these n-order sets is found by subtracting the value 1 from the normalized distance.
6. Finally, the total similarity between two paths will be the sum of similarities of all n-order sets, each one associated with a weighting factor which is usually found by learning.

C. Implementation

Here as an example we took a Temperature monitoring system and finding out the optimal test sequence for it.

The code is provided in Appendix - A

TABLE 1
The generated paths

Shortest Path	Cost	Included Edges	Excluded Edges	Other Paths
s1->s3->s5	8	None	None	
		None	<s5,s6>	s1,s3,s4,s6 = 9
		<s5,s6>	<s3,s5>	s1, s2,s3,s6=12
		<s3,s5><s5, s6>	<s1,s3>	s1,s2,s3,s5,s6=14
s1->s3->s4->s6	9	None	<s5,s6>	
		None	<s4,s6><s5,s6>	∞
		<s4,s6>	<s3,s4><S5,S6>	s1,s2,s4,s6=13
		<s3,s4><s4,s6>	<s1,s3><s5,s6>	s1,s2,s3,s4,s6=15
s1->s2->s5->s6	12	<s5,s6>	<s3,s5>	
		<s5,s6>	<s2,s5><s3,s5>	s1,s3,s4,s5,s6=16
		<s2,s5><s5,s6>	<s1,s2><s3,s5>	∞
……				

The shortest and optimal paths are:

Path	Cost
• s1→s3→s5→s6 -	8
• s1→s3→s4→s6 –	9
• s1→s2→s5→s6 –	12
• s1→s2→s4→s6 –	13
• s1→s2→s3→s5→s6 –	14
• s1→s2→s3→s4→s6 –	15
• s1→s3→s4→s5→s6 –	16

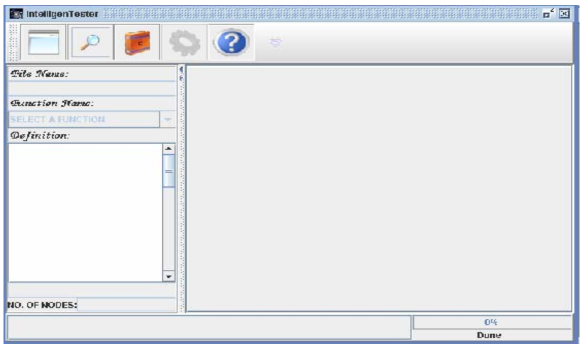


Fig 4. Intelligentester – Window

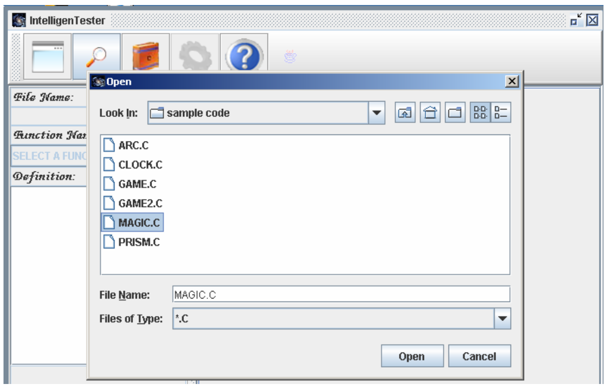
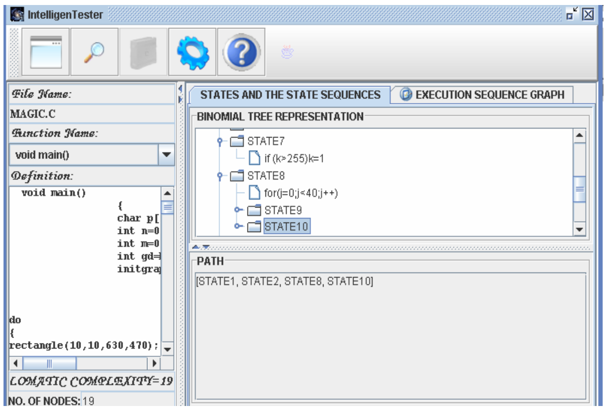


Fig 5. SUT Reader Agent Invocation



```
void main()
{
char p[
int n=0
int m=0
int gd=
initgra

do
{
rectangle(10,10,630,470);
```

```
[STATE1, STATE2, STATE8, STATE10]
```

Fig 6. States and State Sequences Generation for the SUT

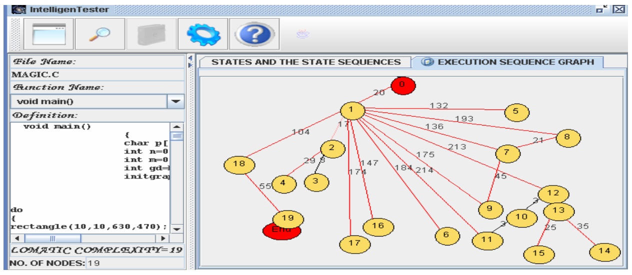
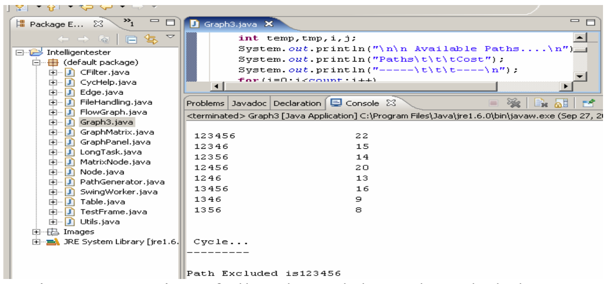


Fig 7. Graph representation of the SUT



```
int temp,tmp,i,j;
System.out.println("\n\n Available Paths....\n");
System.out.println("Paths\t\t\tCost");
System.out.println("-----\t\t\t----\n");
for(i=0;i<count;i++)
```

```
<terminated> Graph3 [Java Application] C:\Program Files\Java\jre1.6.0\bin\javaw.exe (Sep 27, 2

123456              22
12346               15
12356               14
12456               20
1246                13
13456               16
1346                9
1356                8


Cycle...
---------

Path Excluded is123456
```

Fig.8. Generation of all paths and the path excluded 123456

V. Test Case Optimization - ITOA

A. HGA – Introduction

The Hybrid Genetic Algorithms (HGA) are also called as Memetic algorithms [30]. It is a population-based approach for heuristic search in optimization problems. They are more efficient than SGAs in which a local search algorithm is also included so that, the final result will be having global optima. In HGA, Genetic Algorithm is used for attaining global optima and local search algorithms are used for local optima.

B. ITOA Architecture

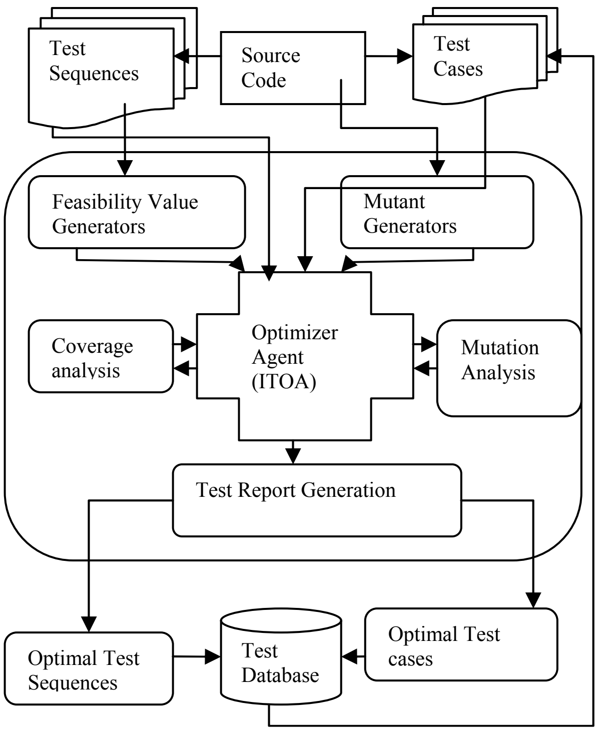


Fig 9. Intelligent Test Case Optimization Framework

C. Generation of Test cases using HGA

All the genetic algorithms consist of the following main components:

- Chromosomal Representation
- Initial Population
- Fitness Evaluation
- Selection
- Cross Over and Mutation

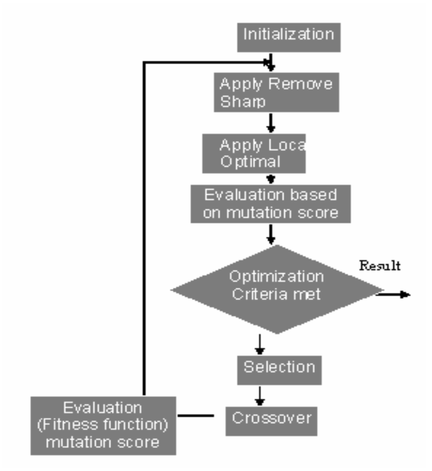

Initialization
Apply Remove Sharp
Apply Local Optimal
Evaluation based on mutation score
Optimization Criteria met
Result
Selection
Crossover
Evaluation (Fitness function) mutation score

Fig. 10. HGA Algorithm

The modified algorithm works as below:

Step 1 :

Initialize population randomly

Step 2 :

- Apply RemoveSharp algorithm to all Test cases in the initial population
- Apply LocalOpt algorithm to all Test cases in the initial population

Step 3 :

- Select two parents randomly
- Apply Crossover between parents and generate an offspring
- Apply RemoveSharp algorithm to offspring [25]
- Apply LocalOpt algorithm to offspring [25]
- If (Mutation Score(offspring) > Mutation Score(any one of the parents)) then replace the weaker parent by the offspring

Step 4 :

Mutate any randomly selected Test case from population

Step 5 :

Repeat steps 3 and 4 until end of specified number of iterations.

1. RemoveSharp Algorithm

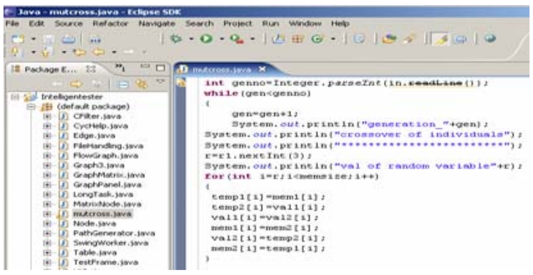
Mutation Score of all the offspring's produced by the n point crossover is calculated. The offspring, which are leaving the more number of mutants in survival, will be deleted from the memory. That is the test cases having very less mutation score will be removed. Test cases having higher mutation score will be memorized.

2. LocalOpt Algorithm

At every generation of offspring by the n point crossover, the offspring, which is having highest mutation score, is selected as local optima. This local optimal solution is compared against the parents. If any of the offspring is better than the any one of the parent then the weakest parent will be replaced by the optimal offspring.

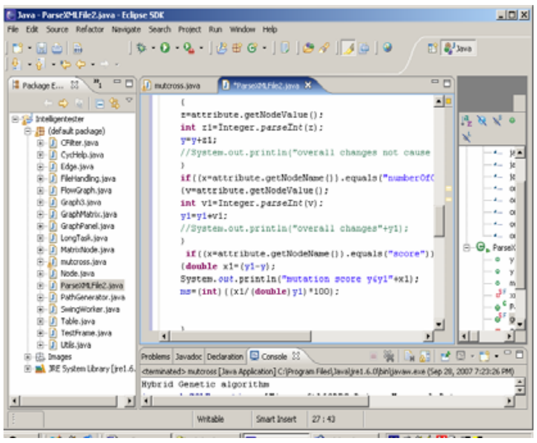
3. Code for HGA

The code was written and executed under Eclipse SDK 3.1 and the mutation score is derived using Jester tool [24]. The output generated by Jester is an XML file, which was then parsed to take the mutation score of each test case. This was compared against the test cases generated by SGA**.** Part of HGA code is given in Appendix-B. Part of the XML parsing and optimal test case selection code is given in Appendix C.



```
int genno=Integer.parseInt(in.readLine());
while(gen<genno)
{
     gen=gen+1;
     System.out.println("generation_"+gen);
System.out.println("crossover of individuals");
System.out.println("**************************");
r=r1.nextInt(3);
System.out.println("val of random variable"+r);
for(int i=r;i<memsize;i++)
{
 temp1[i]=mem1[i];
 temp2[i]=val1[i];
 val1[i]=val2[i];
 mem1[i]=mem2[i];
 val2[i]=temp2[i];
 mem2[i]=temp1[i];
}
```

Fig 11. HGA based generation of individuals using crossover and mutation



```
{
z=attribute.getNodeValue();
int z1=Integer.parseInt(z);
y=y+z1;
//System.out.println("overall changes not cause
}
if((x=attribute.getNodeName()).equals("numberOf(
{v=attribute.getNodeValue();
int v1=Integer.parseInt(v);
y1=y1+v1;
//System.out.println("overall changes"+y1);
}
 if((x=attribute.getNodeName()).equals("score"))
{double x1=(y1-y);
System.out.println("mutation score y&y1"+x1);
ms=(int)((x1/(double)y1)*100);
```

```
<terminated> mutcross [Java Application] C:\Program Files\Java\jre1.6.0\bin\javaw.exe (Sep 28, 2007 7:23:26 PM)
Hybrid Genetic algorithm
```

Fig 12. Mutation Score based Optimized test case generation by XML Parsing

VI. Comparison Charts

Our Test Sequence optimization agent ISA's performance was compared against Ant Colony Optimization based test sequence optimization[10]. The result was shown in fig.13 and fig.14.

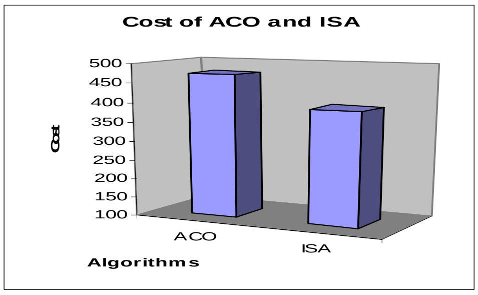


Fig 13. Total Test Sequence Execution cost based on cumulative edge cost

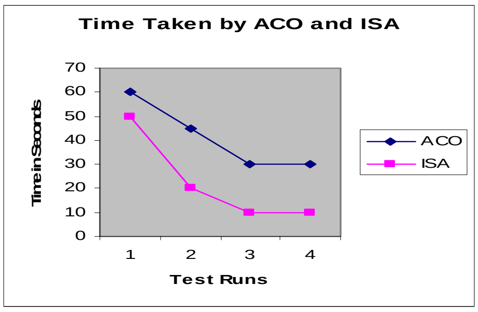


Fig 14. Time taken by ACO and ISA

Our HGA based Test Case Optimization agent ITOA's performance was compared against Genetic Algorithm based Optimized test cases. And the result was shown in Fig.15.

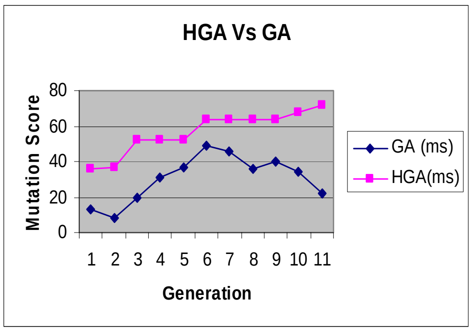


Fig 15. Comparison of HGA based ITOA and GA based on Mutation Score of each test case

VII. Conclusion and Future Work

Our approach has proved to be an optimal one both in terms of time and the total cost needed to finish up the execution of each of the test sequence. Also, our agent has identified the optimal test cases by finding the fitness value as the mutation score. Although our approach includes more number of optimality checks, which may consume time initially, once the agent has learnt the SUT, it will take only less time to further generate the test sequences and test cases during the development of the next build in the SUT.

Appendix – A

```
Class Intelligent_TestSeq_Optimizer
{public void read_SUT()
public void solveProblem(SUT)
{IntelligentSearchAgent ISA = new
IntelligentSearchAgent();
}}
//Inside IntelligentSearchAgent class,
we have the following methods:
Class IntelligentSeaerchAgent
{public IntelligentSearchAgent()
{Blackboard.invloke();
Knowledgesource.reset();
While((!controller.isSolved) ||
(controller.unabletoProceed()))
    Controller.processnextinference();
If (blackboard.isSolved()
    Blackboard.disp_opt_seq();
}}
Class Controller
{public void reset();
public void connect(KnowledgeSource);
public void addinfernce(KnowledgeSource);
public void removeinfere(KnowledgeSource);
public void processnextinference();
public void sort();
public    ndivid isSolved();
public    ndivid  UnableToProceed();
}
…….
Void sort()
{…..}
```

Appendix – B

```
// HGA Code
try       {
            System.out.println("Hybrid Genetic algorithm");
//read initial population and store in database
// Read the next individual's gene value
// Insert first gene values into the data base
//Insert second gene values to the data base
….. //***********Crossover****************
System.out.println("enter the number of generations want
to produce");
       int genno=Integer.parseInt(in.readLine());
       while(gen<genno)
       {  gen=gen+1;
          System.out.println(„generation_"+gen);
       System.out.println("crossover of individuals");
       r=r1.nextInt(3);
       System.out.println("val of random variable"+r);
 //********************Mutation********
 System.out.println("mutation position ,mutation value:
"+mutpos+mutval);                 val1[mutpos]=mutval;
val2[mutpos]=mutval;…..
// End of File
```

APPENDIX – C

```
// Mutation score based test case optimality is done by
parsing the XML code which was produced at the end of
mutation analysis derived from Jester tool
public class ParseXMLFile2 {
  public int y,y1=0,ms=0;
   private final static String xmlFileName =
"c:/rev/jester137/jesterReport.xml";
  public ParseXMLFile2() {
 Document doc = parseFile(xmlFileName);
Node root = doc.getDocumentElement();
writeDocumentToOutput(root,0);
  }
…………
```

REFERENCES

[1] D.Jeya Mala, Dr.V.Mohan, "Intelligent Test Sequence Optimization using Graph Based Searching Technique", Proc. ICISTM, ISBN No. 81-8424-182-8 , pp.10-19, 2007

[2] Briand, L. C.,"On the many ways Software Engineering can benefit from Knowledge Engineering", Proc. 14th SEKE, Italy, pp. 3-6, 2002.

[3] Dorigo M., Maniezzo, V., Colorni, A., "The Ant System: Optimization by a Colony of Cooperating Agents", *IEEE Transactions on Systems, Man, and Cybernetics-Part B,* Vol. 26, No.1, pp.29-41, 1996.

[4] Horgan, J., London, S., and Lyu, M., "Achieving Software Quality with Testing Coverage Measures", *IEEE Computer*, Vol. 27 No.9 pp. 60-69, 1994.

[5] Kit, Edward, "Software testing in the real world – improving the process", Addison-Wesley, 1995

[6] Howe, A. E., Mayrhauser A. V., and Mraz, R. T., "Test Case Generation as an AI Planning Problem", Automated Software Engineering, Vol. 4, pp 77-106, 1997.

[7] Li, H., Lam, C.P., "Optimization of State-based Test Suites for Software Systems: An Evolutionary Approach", International Journal of Computer & Information Science, Vol. 5, No. 3, pp. 212-223, 2004.

[8] McMinn, P., "Search-based Software Test Data Generation: A Survey", Software Testing, Verification and Reliability, Vol.14, No. 2, pp. 105- 156, 2004.

[9] McMinn, P., Holcombe, M., "The State Problem for Evolutionary Testing", Proc. GECCO 2003, LNCS Vol. 2724, pp. 2488-2500, Springer Verlag, 2003.

[10] Huaizhong Li and C.Peng Lam, "Software Test Data Generation using Ant Colony Optimization", Transactions on Engineering, Computing and Technology, 2004, ISSN 1305-5313.

[11] Pedrycz, W., Peters, J. F., "Computational Intelligence in Software Engineering", World Scientific Publishers, 1998.

[12] Tracey, N., Clark, N., .Mander K., and McDermid, N., "A Search Based Automated Test Data Generation Framework for Safety Critical Systems", in Systems Engineering for Business Process Change (New Directions), Henderson P., Editor, Springer Verlag, 2002.

[13] Stuart Russel, Peter Norvig "Artificial Intelligence: A modern approach", 1995, Prentice-Hall Inc.

[14] Danny B. Lange and Mitsuru Oshima, "Java Agent API: Programming and Deploying Agents with Java", Addison-Wesley, 1997

[15] "Intelligent Agents", Centre for Performance Technology, Florida State University, http://www.cpt.fsu.edu

[16] Gilles Brassard, Paul Bratley, "Fundamentals of Algorithms", Prentice Hall of India, 2005.

[17] Ellis Horowitz, Sartaj Sahni, "Fundamentals of Data Structures", Galgotia Book source

[18] Benoit baurdy , Frank fleurey, Jean marc and Yves Le Traon, "Automatic test case optimization: A Bacteriologic algorithm", Mar/Apr 2005 IEEE software,pp-76-81.

[19] Roy P. Pargas, Marry John Harrold, Robert R.Perk, "Test data generation using genetic algorithm" Journal of Software testing, verification and reliability, 1999,pp-1-19.

[20] Srinivasan Desikan, Gopalaswamy Ramesh, "Software testing principles and practices",Pearson 2002 edition.

[21] Paolo Tonella, "Evolutionary Testing of classes", ISSTA-2004,July 11-14.

[22] Andrew Baresel, Harman Sthamer, Michel Schmidt, "Fitness function design to improve evolutionary testing", SBSE-2001.

[23] Roger S.Pressman, "Software Engineering-A practitioner's approach",McGraw.Hill. International sixth edition.

[24] www.jester.com.

[25] G.Andal Jayalakshmi, S.Sathiamoorthy ,R.Rajaram, ,"A Hybrid Genetic Algorithm- A new approach to solve traveling salesman problem", Proceedings of DETC'02 ASME 2002 Design Engineering Technical Conferences and Computers and Information in Engineering Conference.

[26] "EvoTest : Test Case Generation using Genetic Programming and Software Analysis"- Thesis by Arjan Seesing, Delft University of Technology.

[27] Lin Padgham, Micheal Winikoff, "Prometheus – A Methodology for Developing Intelligent Agents", RMT University, http://www.agent-software.com.

[28] Jeromy S.Bradbury, "Using Mutation for the Assessment and optimization of tests and properties", ACM, 2006.

[29] Vincent Kelner, Florin Capitanescu, Oliver Leonard and Louis Wehenkel, "An Hybrid Optimization Technique coupling Evolutionary and Local Search Algorithms", 2004

[30] Natalio Krasnogor and Jim Smith, "A Tutorial on Competent Memetic Algorithms: Model, Taxonomy and Design Issues", IEEE Transactions on Evolutionary Computation, Vol.ANo.B.CCC200D, 2005.

[31] Louire Williams, "Mutation Testing", NC State University, Offutt & Untch, Mutation 2000: Uniting the Orthogonal, 2004.

D .Jeya Mala – received her Masters degree in Computer Applications and Philosophy in Computer Science from the Madurai Kamaraj University, Tamil Nadu, India. Currently she is a Ph.D., research scholar of Anna University, Chennai. Her papers were published in International Conferences like ICISTM, ICRSQE, ICAC and many national conferences. Also, her papers were selected in many international journals. Her research interests include Artificial Intelligence, Software Engineering, Software Testing, SOA and Web Development.

Dr.V.Mohan - received Doctoral degree in Mathematics from Madurai Kamaraj University, Tamil Nadu, India. He is currently working as a Dean – Planning and Administration. His research interests include Graph Theory, Artificial Intelligence and Finite State Automata. His papers were published in various national and international Journals and Conferences.