OptiTest: Optimizing Test Case Using Hybrid Intelligence

[1] Mohammed Al-Fayoumi, [2]P .Mahanti and [3]Soumya Banerjee

Abstract- **This paper deals with the identification of best and optimized test cases in program components and software artifacts. Our purpose is to simulate the model on a sample software program component and evaluate the efficacy and correctness of the code through set of test cases.**

Index Terms— **Software Testing, Test case, Hybrid Intelligence, Model Simulation.**

I. INTRODUCTION

Many accomplished researchers have claimed, "Programmers love writing tests" **[9],** they explore their confidence level in their written code when it passes through their tests. Undoubtedly, software testing remains the primary technique used to gain users confidence and trust in the software product. It has also been observed that on software testing usually accounts for about 50% costs **[6]** of software development. The application of artificial intelligence technique in software engineering and testing is an emerging area of research that brings about the cross fertilization of ideas across two domains. A number of published works, for example **[12]** have begun to search the effective use of AI for SE relate activities, which are inherently knowledge intensive, and human centered. Similarly, the prominent uses of AI in software testing have also been reported in some significant works through genetic algorithm, AI planner simulated annealing and even by ACO**[1][10][18**]. Generating a set of basic test cases might be easier to implement, improving the test quality and efficacy require substantial effort and investment. The test cases that software tester generally provide easily cover 50-70% of introduced faults. But improving the score up to 90-100% is complex, time consuming and hence proved to be an expansive method. Therefore the optimization of test cases is required and practically important. This process could be automated and less time consuming with perfection through hybrid intelligent technique. Improving the quality of generated test cases (especially in case of unit testing) automatically is a non-linear optimization problem**.** In order to tackle this problem, we have developed an algorithm called as *OptiTest* based on hybrid intelligence. The genesis of the algorithm is the implementation of ant colony and its internal pheromone distribution across the generated test graph. On the other hand the algorithm also incorporates another popular intelligent tool commonly known as Rough Set. From the perspective of search-based software engineering, the rough set based rule would like to denote the completion of search for optimized test case. This novel hybrid metaphor has been generated test graph. On the other hand the algorithm also incorporates another popular intelligent tool commonly known as Rough Set. From the perspective of search-based software engineering, the rough set based rule would like to denote the completion of search for optimized test cases.

This novel hybrid metaphor has been applied on a test source code of C # in .Net framework. The rest of the paper is organized as follows: Section II explores a brief outline of evolutionary test and search-based software testing scenario. The background of AI based techniques used in software testing and related works have been detailed in section III and its subsection. Section IV describes the proposed model followed by section V of C # code under test. Section VI provides different similar works followed by conclusions in section VII wherein we have presented the extension and future scope of this model.

II. EVOLUTIONARY TESTING OF CLASSES AND SEARCH BASED TESTING PARADIGM

Automated test case generation for object-oriented program has always become challenging part of software testing. The test case for procedure consists of a sequence of input values, to be passed to the procedure upon execution. Therefore, test cases for class method must also account for the state of object on which the method execution is issued. Thus practically a test case for a class method includes the criteria of an object, optionally the change of its internal state and finally invocation of the

[1] Professor ,Computer Science at the Middle East University for Graduate Studies, Amman ,Jordan(email:fayoumi99@yahoo.com)

[2] Professor, Computer Science, University of New Brunswick, Canada, (email:pmahanti@unbsj.ca.), Corresponding author.

[3] Assistant Professor, Birla Institute of Technology, Mesra, India.

method with proper input value. In this context unit testing (the fault is injected into single class) of such a class should work upon all associated parameters. The steps of unit testing are as follows **[21]:**

- An object of the Class under Test (CUT) is created using one of the available constructors.
- A sequence of zero or more methods is invoked on such an object to bring it to a proper state.
- The method currently under test is invoked.
- The final state reached by the object being tested is examined to assess the result of the test case.

In a nut shell, unit testing of a class consists of a sequence of object creations, method invocations and then final method invocation under test.
For example, if we are testing method m of class A, a test case may be:

```
A a = new A ( );
B b = new B (  );
b.f(2);
a.m(5, b);
```

Here, the second object creation is necessary, because the second parameter of m is an object of class B. Invocation of f on b aims at changing the state of b before passing it to m. In this context of this proposal we would also like to brief about the search based test data generation technique, which also largely influence this implementation. The search based test data generation requires some basic concepts as prerequisites, which starts from control flow graph **[18]**. A control flow graph (CFG) of a program is a directed graph G = (N, E, s, e) where N is a set of nodes, E is a set of edges, and s and e are unique entry and exit nodes to the graph. Each node n ∈ N corresponds to a statement in the program, with each edge e = (ni, nj) ∈E representing a transfer of control from node ni to nj. Nodes corresponding to decision statements (for example an if or while statement) are referred to as *branching node.* In the Figure 1 nodes 1, 2, and 3 are all branching nodes. Outgoing edges from these nodes are referred to as branches. The branch executed when the condition at the branching node is true is referred to as the true branch. Conversely, the branch executed when the condition is false is referred to as the false branch. The predicate determining whether a branch is taken is referred to as a branch predicate. The branch predicate of the true branch from branching Node 1 in the program of Figure 1 is 'a >= b'.

CFG Node

```
(s) void example (int a, int b, int c, int d)
        {
            (1) if (a >= b)
        {
            (2) if (b <= c)
{
            (3) if (c == d)
{
    // T.....
```

Fig 1: Code Segment for CFG

The comprehensive suit of unit testing is well supported successfully by Evolutionary algorithms for procedural software (**[19, 7],** referred to as *conventional evolutionary testing*). Even the application of metaheuristic search techniques to test data generation is a possibility, which offers much promise for these different types of software testing problems. Metaheuristic search techniques are high-level frameworks, which utilize heuristics in order to find solutions to combinatorial problems at a reasonable computational cost. Such a problem may have been classified as NP-complete or NP-hard, or be a problem for which a polynomial time algorithm is known to exist.

III SOFTWARE TESTING AND AI

The application of artificial intelligence methodologies in software testing have been reported in several accomplished works. The varieties of AI based tools are applied for test data generation, search, optimization and coverage analysis and test management .Most commonly applied tool is genetic programming **[20].**
Very recently couples of noted works of Baudry et al. exhibit a new dimension of testing through their simulated bacteriological adoption algorithm **[3] [4] [5].**
which significantly contributed to formulate certain basic framework for this type of testing paradigm by proposing different object specific mutation operators dedicated to Java language **[21] [8]**. The concept of interface testing and validation has already been used to test EJB components **[2].** All these works actually map the problem for test data generation to the problem of minimization and study Genetic Algorithm to tackle this minimization problem.
They also tested their schemes to .Net components **[4].** In practical testing scenario, it's difficult to generate test data manually, as it makes the code more and more error prone. Therefore automated test data generation **[16]** has become authentic approach for software testing. The various forms of this technique have been found in the local search **[19] [20]** (for searching structural test data generation) with simulated annealing **[15] [21]** and other evolutionary algorithms. The major difference between all these work and the proposal presented in this paper is that they are concerned in generating the scalar data, whereas method calling and argument passing in the intermediate control flow dependency graph is the basis of this paper. The other specialty of the proposal is to create a novel hybrid framework inspired by natural agent ant, which lives in a colony and on the other hand rough set theory, which work under uncertainty and in imprecise conditions. The distributed coordination mechanism of ant agents

through a chemical marker called as pheromone also able to notify and mark best or good path of choice. This path is shortest by default. Rough Set Theory offers the heuristic function to measure the quality of a single subset of test case. Rough set is also used to stop the searching process of test cases if best or optimized test case is found. The next subsection will present some basics on these two methodologies to understand the proposed algorithm *OptiTest.*

A. Background of Ant colony and Rough Set Theory

Ant Colony Optimization (ACO) **[13] [14]** is a recently proposed meta-heuristic approach for solving hard combinatorial optimization problems. The inspiring source of ACO is the pheromone trail laying and following behavior of real ants, which use pheromones as a communication medium. In analogy to the biological example, ACO is based on the indirect communication of a colony of simple agents, called (artificial) ants, mediated by (artificial) pheromone trails. The pheromone trails in ACO serve as distributed, numerical information, which the ants use to probabilistically construct solutions to the problem being solved, and which the ants adapt during the algorithm's execution to reflect their search experience. Rough set theory **[23]** is an extension of conventional set theory that supports approximations in decision-making. The rough set itself is the approximation of a vague concept (set) by a pair of precise concepts. Here the quality of test case is measured and the process for searching "best" quality test case is stopped by rough set constructs.

IV. PROPOSED MODEL –*OPTITEST*

The proposed algorithm envisages the modeling of the unit test mentioned in the previous section **(refer section II)** for object oriented source code. The algorithm takes as input initial set of test cases and it outputs a "good" or "optimized" set of test cases. The process goes on incrementally as *test pheromone* corresponds to a test case. We consider a C# parser, the input data here is a source file that is parsed to build a syntactic tree. Subsequently, the source code is interpreted in term of CFG, which is a directed graph. There are several issues need to be addressed prior to deploy ACO:

- Interpretation of testing problem with decomposable graph.
- A heuristic measure for measuring the "goodness" of path through the graph.
- A rough set based rule for stopping the search.
- A transition rule for determining the probability of an ant traversing from one node in the graph to the next.
- Update matrix for pheromone deposition by ant colony on a particular edge of test graph.

High-level description of proposed algorithm --- *OptiTest*

Begin Algorithm GenerateTestCases (Class Under Test - CUT: Class)
/* Generate Test Case*/
Input: CUT
Output: Deterministic Best Test Case

/*Initialize Parameters for Ant Colony and Rough set Paradigm*/
Set t: = 0 **/* t is the time counter */**
Set NC: = 0 **/* NC is the cycle counter*/**
for every edge (i, j) set an initial value τ_{ij} = C for trail intensity and $\Delta \tau_{ij}$ = 0
Bool TestPheromoneValue= FALSE;
Tabu = Empty;
Place the m ants on the n nodes

P, Q equivalent Relations over Test case $\bigcup$

/ * Stopping Criteria of search defined by Rough set */
Testgoals:= the set of coverage goals
Failures: = empty set
Identify test cluster for CUT
/*Initialize Parameters for generating Test case*/
Device Source code for Test Cluster
/*Example Test: for C# Code to be parsed to .Net platform*/
Accumulate test Goals for CUT
Generate function set for Test Cluster
for each test goal tg in TG
/* Loop for generating normal test goals*/
modify function test for TG
end for
Create initial di- graph based test cases
for cycle =1 to ncycles do
for ant =1 to n ants do
Select the vertex with the lowest pheromone level from the current vertex
If vertices vi and vj shares the same lowest pheromone level but if TG (Vi) = =0 and TG (Vj) =1
Select vi
else
Select randomly any vertex
end if
ant clears its *recentlyvisited* tabu list
/* no test Goal in Tabu*/
UpdatePheromone (τ, ρ)
/*Pheromone Update Rule*/
update pheromone trails by applying the rule
$\tau_{ij}(t) \rightarrow (1-\rho).\tau_{ij}(t) + \Delta\tau_{ij}(t)$ where

$$\Delta \tau_{ij}(t) = \begin{Bmatrix} 1/c(\pi+) \\ 0 \end{Bmatrix} \text{ otherwise}$$

end for
Set TestPheromoneValue = TRUE;

$$p = \frac{\tau(r,u)^{\alpha} * \eta(r,u)^{\beta}}{\sum_{k} \tau(r,u)^{\alpha} * \eta(r,u)^{\beta}}$$

According to ant movement rule:
/* $\tau(r,u)$ is the intensity of pheromone on the edge between the node position r and u, α and β represents a weight for the pheromone and weight for the heuristic of $\tau(r,u)$ */
while *Testgoals* is nonempty **do**
Select and remove *goal* from *goals*
Call PheromoneUpdate( ) to choose Best new test case path to discharge *goal.*
if successful **then**
Select and remove from *goals* any that are discharged by the test case
remaining := empty set
while *goals* is nonempty **do**
Remove *goal* from *goals*
Call PheromoneUpdate( )to **extend** test case to discharge *goal*
if successful **then**
remove from *goals failures*, and *remaining* any goals discharged by extended test case
else add *goal* to *remaining*
endif
endwhile
goals := *remaining*
Set P and Q in test case digraph such that $P \Rightarrow Q$ /* **when to terminate Search for best test case*/**
And the dependency degree k by P, Q is related is $0 \le k \le 1$ denoted $P \Rightarrow kQ$
if
$k = \gamma p(Q) = Positivep(Q)/\cup$ **/* Rough Test Rule*/**
else
k =1
Q totally depends on P
else
Q doesn't depend on P
Output test case
else add *goal* to *failures*
endif
endwhile

// Source Code in C#

```
public static void BubbleSort(String
arr)
{
      int i,temp, j, arr1;
      int [ ] array = new int[5];
      arr1 = interface.parse (arr) ;
for (i=0;i<5; i++)
      arr[i] =
      interface.parse(system.console.R
      eadLine ());
System.Console.WriteLine("Sorted is::
")
for(i=0;i<4;i++)
  {
      for(j=0;j<4-i;j++)
      {
            if(array[j]>array[j+1])
            {
                  temp=array[j];
                  array[j]=array[j+1];
                  array[j+1]=temp;
            }
      }
  }
System.Console.WriteLine("the value
passed to funtion is "+ arr1);
for(i=0;i<5;i++)
   System.Console.WriteLine(array[i]);
   System.Console.WriteLine("value
returned to bubblesort is");
}

Static void Main(string[] args)
{
String hj ="hello";
for(int i=0;i<2;i++)
BubbleSort(args [i]);
temp1("hi");
System.Console.WriteLine("Bublesort
begins again");
BubbleSort(System.Console.ReadLine());
temp1(hj); temp1(123);
System.Console.WriteLine("exit from
main");

System.Console.Read();
}
}
```

V. TEST CODE ANALYSIS AND PERFORMANCE OF THE MODEL

The Following code segment and the conceptual dependency graph for C# have been presented. The code describes common bubble sort mechanism and parameter passing.

The sample erroneous C# code has been prepared for the testing the proposed model *OptiTest.* The algorithm is incremental in nature. The code provides its dependency graph describing the relation between the caller and callee method in he code. The certain set of test cases have been prepared for the source code (refer the entire test case in table 1, given below). The ant agent traverse those set of test cases employed on dotted erroneous path. We denote the input domain as TG or Test Goal. Each iteration of input process involves some basic functions. The ant agent has the sense of Test Pheromone value. The distribution function of pheromone matrix can be given as:

best Test Goal Path: $2^{TG} \rightarrow N^{+}$, where N is a real number, comprises a positive pheromone value that is capable of identifying the quality of a set of test case as per its Boolean objective i.e. TRUE or FALSE (the other parameters of pheromone update is set accordingly). The relational of two test cases denoted by P and Q (their dependency k has been defined) accepts the test case as an input and determine its appropriateness in the context of error trace and test the threshold value with their dependency value k. If it exceeds k then the algorithm outputs that set of test case(s) as the "best "or "good" test case achieved so far(marked as green label in Table 1).

	Variable	**Value**	**Return**	**Remark about Test Case**
Bubblesort→BubbleSort(args[i])	args[i] args[i] **args[i]**	123 args[i] **+1.AB3**	123 error **error**	Best test case
System.Console.ReadLine ( )	buffer buffer **buffer**	12568 @AB# **+/-23**	12568 error **error**	No suitable test case
temp1( )	***name***	***123***	error	Best test case
int. parse( )	buffer buffer **buffer**	14 +2.AB **+*AB45**	14 error **error**	No suitable test case

Table 1: Error Identification and Test Case

Note: The work considers 4 basic methods written in #C code(the code demonstrates popular Buuble Sort technique, and intentionally the code is kept erroneous for testing), where each have been tested by passing some assumed test value and observed against the return value and path of the given test value. This method of passing and checking the return value of the method could identify the best test value and thus best test case (marked as bold). The 3rd case in the table 1 describes the single variable, hence evaluates most simple and non complex return value as error (thus it's also the best test case applied on a single variable marked as bold italics). According to the proposed algorithm *OptiTest,* the distribution and search of best test case value has been done through Ant colony pheromone matrix and once the search of best test value is achieved, the search terminates through Rough set.

The dependency relation of P and Q is defined as rough set value for *OptiTest* basically set for feature selection of best test case set identified after certain iteration. Hence, rough set is here used for forming stopping criteria rule in the proposed model. The return value of rough set is definitely the best test goal path or TG, which is $^{+}$ or a Boolean value of pheromone distribution. The model embeds a test case grammar, syntax tree manager on which the model is applied. We have initialized several parameters in *OptiTest* to use the hybrid metaphor, he most relevant is the setting of threshold and the other is the size of test case (assuming the grammar of test case and syntax tree is available). Apparently it may appear that the rule of generating test case is applied, but in reality we assume that all complexity of the test case of our code fixed.

VI. SIMILAR WORKS

The need for testing-for-diagnosis strategies has been identified for a long time, but there is always a dilemma between a reduced testing effort (with as few test cases as possible) and the diagnosis accuracy (that needs as much test cases as possible to get more information). There are reported recent published works which filter the different test cases and finally is able to select the optimized one **[22].** Certain interesting observations have been inferred from research where the optimization is done through use cases driven test cases for embedded object oriented software **[17].** Even optimization approach of test cases have become so popular that different works adopt practical applications of the optimization model for system test planning **[11].**

VII. CONCLUSION AND FURTHER SCOPE

The present work evaluated the source code (in C#) with the help of insect ant and precision is handled by rough set. It has been observed that certain test path on dependency graph has been passed thorough and some are failed and even certain paths are more trusted depending on the goal set by each test case. The labeling of the path is done by Test Pheromone of ant agent and trust or belief of any test path depends not only on statistical value but also to pinpoint faulty statements in a program. The promising scope of the present work may be in the form of different ant colony based approach and the setting of the parameters of this algorithm. Initial test pheromone value and feature for particular test case can also be an interesting proposition on the performance issue of similar algorithm.

REFERENCES

[1]. A.E. Howe, A.V.Mayrhauser Mraz, R.T., "Test case generation as an AI planning problem", Automated Software Engineering vol. 4, pp 77-106, 1997.

[2]. Benoit Baudry, Franck Fleurey, Jean-Marc J´ez´equel and Yves Le Traon ," From genetic to bacteriological algorithms for mutation-based testing Software Testing Verification And Reliability" *Softw. Test. Verif. Reliab.* 2005; 15: pp.73–96 published online 4th January 2005 available http:// www.interscience.wiley.com.

[3]. B. Baudry, F. Fleurey, Y. Le Traon, and J.-M. Jézéquel. "An Original Approach for Automatic Test Cases Optimization: a Bacteriologic Algorithm". IEEE Software 22(2): pp. 76-82, March 2005.

[4]. B Baudry, F. Fleurey, Y. Le Traon, and J.-M. Jézéquel, "Genes and Bacteria for Automatic Test Cases Optimization in the .NET Environment". In proceedings of ISSRE'02 (Int. Symposium on Software Reliability Engineering), Annapolis, MD, USA, pp. 195-206, November 2002.

[5]. B Baudry, F. Fleurey, Y. Le Traon," Improving Test Suites for Efficient Fault Localization" In Proc. *ICSE'06*, May 20–28, 2006, Shanghai, China. Copyright 2006 ACM 1-59593-085-/06/0005, pp 82-91.

[6]. G.J. Myers, *The art of software testing, Wiley*, 1979.

[7]. H. Sthamer, J. Wegener, and A. Baresel." Using evolutionary testing to improve efficiency and quality in software testing." In *Proceedings of the 2nd Asia-Pacific Conference on Software Testing Analysis and Review (AsiaSTAR)*. 22-24th July, 2002.

[8]. H Yoon ,B Choi ," Effective test case selection for component customization and its application to Enterprise JavaBeans" *Software Testing, Verification and Reliability* 2004; 14(1):pp.227–247.

[9]. K. Beck, E. Gamma. "Test-infected: Programmers love writing tests." **3**(7), pp. 37–50, *Java Report* 1998.

[10]. K. Doerner, W.J. Gutjahr, "Extracting test sequences from a Markov software usage model by ACO", LNCS, vol. 2724, pp-2465-2476, Springer Verlag, 2003.

[11]. K. Chari, A Hevner, "System Test planning of Software: An Optimization Approach" in IEEE Transactions on Software Engineering, vol. 32, no. 7, July 2006, pp. 503-509.

[12]. L.C. Briland.,"On the many ways Software Engineering can benefit from knowledge engineering", Proc. 14th SEKE, Italy, pp3-6, 2002.

[13]. M. Dorigo and G. Di Caro" The Ant Colony Optimization meta-heuristic" *In D. Corne, M. Dorigo, and F. Glover, editors, New Ideas in Optimization,* pp. 11–32. *McGraw Hill, London, UK, 1999.*

[14]. M. Dorigo, G. Di Caro, and L. M. Gambardella. "Ant algorithms for discrete optimization." *Artificial Life, 5(2):*pp.137–172, 1999.

[15]. N. Tracey, J Clark, and K. Mander.,"The way forward for unifying dynamic test-case generation: The optimization-based approach" In Proc. International Workshop on Dependable Computing and Its Applications, pp. 169–180. Computer Science Dept., University of Witwatersrand, South Africa, 1998.

[16]. N. Tracey, J. Clark, K. Mander, and J. McDermid," An automated framework for structural test-data generation" In Proc. International Conference on Automated Software Engineering, pp. 285– 288, Hawaii, USA, IEEE Computer Society Press, 1998.

[17]. Nebut Cle´mentine, F. Fleurey, Y. Le Traon "Automatic Test Generation: A Use Case Driven Approach" in IEEE Transactions on Software Engineering, vol. 32 no.3, pp.140-154, March 2006.

[18]. P. McMinn, Mark Harman, David Binkeley and Paolo Tonella, "The Species per Path Approach to Search Based Test Data Generation", Proc. ISSTA July 17-20, 2006, ACM 2006.

[19]. P. McMinn, M. Holcombe, "The State Problem for Evolutionary Testing", Proc. GECCO 2003, LNCS Vol. 2724. pp. 2488-3500, Springer Verlag, 2003.

[20]. P. McMinn, "Search-based test data generation: A survey", *Journal on Software Testing, Verification and Reliability*, 14(2):105–156, June 2004.

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

[22]. P. Chevalley, "Applying mutation analysis for object-oriented programs using a reflective approach", Proceedings *of the 8th Asia-Pacific Software Engineering Conference*, Macao, China, IEEE Computer Society Press: Los Alamitos, CA, 2001; pp.267–270, December 2001.

[23]. Z. Pawlak,"Rough Sets: Theoretical Aspects of Reasoning About Data" Kluwer Academic Publishing, Dordrecht, 1991.