Optimal Test Sequence Generation in State Based Testing Using Cuckoo Search

Article · January 2012

DOI: 10.4018/jaec.2012070102

Dr. Praveen Ranjan Srivastava
Indian Institute of Management, Rohtak

Optimal Test Sequence Generation in State Based Testing Using Cuckoo Search

Praveen Ranjan Srivastava, Birla Institute of Technology & Science, Pilani, India

Ashish Kumar Singh, Birla Institute of Technology & Science, Pilani, India

Hemraj Kumhar, Birla Institute of Technology & Science, Pilani, India

Mohit Jain, Birla Institute of Technology & Science, Pilani, India

ABSTRACT

The present work describes a method for increasing software testing efficiency by identifying the optimal test sequences in the state machine diagram. The method employs a Meta-heuristic algorithm called Cuckoo Search to investigate best paths in the diagram. It tries to provide a technique for exhaustive coverage with minimal repetition which ensures all transitions coverage and all paths coverage at least once with minimal number of repetitions of states as well as transitions. The algorithm works by maximising an objective function which focuses on most error prone parts of the program so that critical portions can be tested first. State machine diagram is given as input, and Cuckoo Search is performed to generate a list of test sequences as output.

Keywords: *Cuckoo Search, Fitness Function, Levy Flights, Random Walk, Software Testing, State Based Testing, State Model, Test Sequence Generation*

1. INTRODUCTION

For the proper functioning of societies whether national or international, software engineering is the fundamental ingredient. It is almost impossible to run the modern world without software. Software systems are abstract and intangible. Throughout the 1970s and 1980s, a variety of new software engineering techniques and methods (Dijkstra, 1970; Parnas, 1972) were developed, such as structured programming, information hiding and object oriented development (Pressman, 2001).

The increasing use of software in the organisations and the costs associated with a software failure are motivating forces for well-planned software development followed by extensive testing. Generally a software development organisation incurs a cost of 30 to 40 percent in software testing of the total project development effort (Pressman, 2001). Given such a scenario, study of software testing is of utmost importance.

DOI: 10.4018/jaec.2012070102

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test (Karner, 2006). It is an integral part of software quality assurance as it is concerned with finding faults in software. Usually, the objective of software testing is to find minimal set of test cases such that, as many faults as possible are uncovered. As mentioned previously, software testing is a very lengthy process and requires a significant amount of time (Alander, Mantere, & Turunen, 1998).

Over the years many researchers have proposed (Mayers & Glenford, 2004; Edgar, 1989) different methods for software testing. With the increasing popularity of object oriented programming languages, the research in the field of object oriented software testing has been done increasingly. As the size of the commercial softwares is usually large, automated testing tools are required (Srivastava, Ray, & Raghurama, 2009). Although a lot of research has been done in the field of software testing, optimal test sequence generation in an automated way is still an area of concern for such testing tools.

Empirical studies have shown that state based testing methods are more effective than random testing (Offutt et al., 2003). Also, the current research in software testing using Meta-heuristic techniques have shown more promising results than traditional techniques (Doungsa-ard et al., 2005; Li & Lam, 2004; Li, Zhang, & Liu, 2009; Rathore et al., 2011; Srivastava et al., 2010a, 2010b).

In view of the findings the proposed approach is a meta-heuristic method based on state transition diagram of the software under test. A study (Gandomi, Yang, & Alavi, 2011) of various structural optimization problems indicate that cuckoo search algorithm fairs better than other algorithms. Therefore, the proposed approach employs Cuckoo Search algorithm to find the best sequences in a state diagram. In 2009 Xin-She Yang and Suash Deb developed an optimization algorithm named Cuckoo search (CS) (Yang & Deb, 2009), which was inspired by obligate brood parasitism of some Cuckoo species that lay their eggs in the nests of birds of other species. In addition Levy Flight (Leccardi, 2005) distribution is used to emulate the random walk characteristic of the Cuckoo bird. The algorithm uses an objective function which focuses on most error prone parts of the program so that critical portions can be tested first. The approach tries to provide a technique for exhaustive coverage with minimal repetition.

This paper is structured as follows. Section 2 discusses the previous work in the field of software testing. Section 3 describes the Cuckoo Search algorithm and other techniques used in the proposed approach. Section 4 presents proposed approach for test sequence generation. Section 5 gives the experimental research by providing an example. Section 6 describes the analysis of the suggested approach for test sequence generation, and finally Section 7 concludes the paper.

2. BACKGROUND

At a very primary level, software testing can be divided into two broad categories namely white box testing and black box testing (Pressman, 2001). State based testing is a type of black box testing. There has been increasing level of research in the testing of object oriented programs using finite state machines. Test generation methods based on FSMs include “tour” (Naito & Tsunoyama, 1981), the “W-method” (Chow, 1978) and the “Partial W-Method” (Fujiwara et al., 1991). The research paper (Turner & Robson, 1993) draws similarities between finite state automata and state transition diagram and describes a method for testing using state machine diagram. The disadvantage with mentioned approaches is that most of these specification based testing techniques use manual methods that cannot be generalized or automated.

The approach of Delha Sokenou (2006) provides a way for test sequences coverage of state diagrams. The state diagrams are first represented as sequence diagrams and then test sequences are generated. It is assumed that the sequence and state models are consistent.

The events in state diagram are converted to method calls in sequence diagram. One of the problems with the approach is that converting the state diagram into sequence diagram robs the dynamic behavior offered by state diagram. Furthermore, it does not give any priority to parts of the software under test which are considered more error prone.

Considering the difficulties, the current research in software testing has moved towards using meta-heuristic techniques in testing. These techniques have shown more promising results than traditional techniques (Doungsa-ard et al., 2005; Li & Lam, 2004; Li, Zhang, & Liu, 2009; Rathore et al., 2011; Srivastava et al., 2010a, 2010b).

An approach to obtain test sequence based coverage of state machine diagram using genetic algorithms (Doungsa-ard et al., 2005), establishes the criteria of the quality of the test sequences. It states that more the number of transitions fired by the test data being generated, the better is its quality. However, the proposed approach failed to provide complete coverage.

A model proposed by Li et al. (2009) to generate test data, based on ant colony optimization technique along with the branch function technique and problem of local optimization has also been solved. But this approach is applicable only to the numeric data types and the model is not suitable for object oriented programs.

A coverage based approach (Rathore et al., 2011) uses genetic algorithm along with Tabu search on control-dependence graph. The objective function used in the approach uses only the number of common predicates with the target node for finding objective value. This can be seen as a disadvantage since the optimal graph traversal usually depends on number of factors such as distance from initial and final node, criticality etc. which it fails to take into account.

An approach suggesting use of genetic algorithm on hamming distance was presented in Srivastava et al. (2010b). By examining the most critical paths first, it obtains a more effective way to approach testing which in turn helps to refine effort and cost estimation in the testing phase. Nevertheless, the proposed approach has likelihood to reach towards local optima or random points and not to global optima of the problem.

An ant colony optimization approach (Srivastava et al., 2010a) tries to cover most critical nodes in the Markov chain based usage model. The model incorporates factors like cost and criticality of various states in the model. Here although, the most critical nodes have been covered, complete state and transitions coverage has not been achieved.

Another ant colony optimization based algorithm in Li and Lam (2004) tries to find test sequences in a UML state chart diagram. The research provides for all state coverage and a feasible test path but fails to impart optimality to the test sequences. The sequences produced were usually much longer than the shortest possible sequences.

Despite lot of research in this field, efficient test sequence generation has not been possible. The proposed approach tries to resolve the problem towards exhaustive state coverage and optimal sequence generation.

3. CUCKOO SEARCH AND SOFTWARE TESTING

The applications of Cuckoo search into engineering optimization problems have shown its promising efficiency. Furthermore, Cuckoo Search is particularly suitable for large scale problems, as shown in a recent study (Speed, 2010). Using Cuckoo Search in software testing and particularly state based testing can prove to be immensely beneficial.

3.1. Cuckoo Search Algorithm

Cuckoo search is an optimization algorithm developed by Yang and Deb (2009). It is based on the parasitic behaviour of some Cuckoo species that lay their eggs in the nests of host birds of other species. Some of the eggs laid by Cuckoos are detected by host bird of that nest, these alien eggs are either thrown away or the host bird abandons that nest to build a new nest elsewhere. Over the time some Cuckoo species like new world brood-parasitic Tapera have developed capabilities that eggs laid by them are very close to the colours and patterns of the eggs of some selected host bird species (Payne, Sorenson, & Klitz, 2005).

The following representation scheme is chosen by Cuckoo Search algorithm:

Each egg in a nest represents a solution, and a Cuckoo egg represents a new solution. The aim is to use the new and possibly better egg to replace a not-so-good egg of Cuckoo in the nests. However this is the basic case i.e., one cuckoo per nest, but the extent of the approach can be increased by incorporating the property that each nest can have more than one egg which represents a set of solutions.

Cuckoo search is based on three idealized rules:

- Every Cuckoo lays one egg at a time in an arbitrarily selected nest
- The best nests with superior eggs i.e., the cuckoo eggs that are not detected by host bird as alien, will hatch and are carried over to the next generation
- It is possible that the egg laid by Cuckoo is discovered by host bird, the discovered eggs are thrown away by the host bird. Thus, the discovered solutions are removed from future calculations.

Additionally, Yang and Deb (2009) found out that the random-walk style search used in CS should be performed via Levy flights as it is superior to simple random walk and is much closer to original behaviour of Cuckoo.

3.2. Levy Flights

The purpose of Levy flight is to furnish a random walk while levy distribution is responsible for generating the random step length. The levy distribution can be written as

$$Levy \sim u = t^{-\lambda}, \ (1 < \lambda \le 3)$$

It has an infinite variance with an infinite mean. The steps form a random walk process with a power law step length distribution with a heavy tail. Some of the new solutions should be generated by Levy flight around best solution obtained so far, this will speed up the local search. To be safe from the problem of local optimum (Leccardi, 2005), the locations should be far enough from present best solution. This can be achieved by applying far field randomisation to calculate substantial fraction of new solutions.

Levy flight can provide the probable next state to be covered in state diagram. The application of levy flight in the proposed approach is given in Section 4.

3.3. State Based Testing

State diagrams are used to describe the behaviour of a system. State diagrams describe all of the possible states of an object as it transitions from one state to another as events occur with that object. Each state diagram generally portrays objects of the same class and tracks its transition to different states through the system (Blaha, 2011). The UML state diagram notation (Blaha, 2011) has been followed in this approach to model state based diagrams.

The state based testing technique tries to test the software by covering all the transitions and states so that any underlying error in the software can be discovered.

In software testing, state based testing is a commonly used approach. However, state based testing has two limitations associated with it: (1) it may happen that a certain number of test cases are infeasible; (2) to obtain specified test coverage many repetitive test cases have to be generated. As far as we know, no systematic and efficient plan of action has been reported that deals with automatic generation of test cases that are both feasible and efficient (Li & Lam, 2004).

Next section discusses optimal test sequence generation based on Cuckoo search, levy flight and state machine in a single platform.

4. PROPOSED APPROACH

This paper suggests an approach to generate automatic test sequences and provides a solution to cover all transitions. The purpose of Cuckoo Search optimization algorithm is to cover the optimal transitions in the UML state machine diagram at least once.

It tries to provide a technique for exhaustive coverage with minimal repetition which ensures all transitions coverage and all paths coverage. The algorithm works by maximising an objective function which focuses on most error prone parts of the program so that critical portions can be tested first. State machine diagram is given as input, and Cuckoo Search is performed to generate a list of test sequences as output.

The architecture of the testing tool is shown in Figure 1. The input is given in the form of a state diagram which can be designed using UML2 plugin of Eclipse IDE. This diagram is stored as a XML syntax file which can be parsed with a XML parser to represent the state diagram in the form of data structures. Cuckoo search algorithm is applied on these data structures to find out test paths.

Abbreviations used in the algorithm:

- Next state to be covered from present state in the test sequence: Egg
- Number of transitions possible from the node in state diagram (or outbounds): OB
- The tester inputs the value of criticality of state: criticality
- The tester inputs the frequency of occurring a state: frequency
- Distance of the present node from final node: d_{final}
- outbound still to be traversed from the present state: $OB_{\text{remaining}}$
- A stack which holds the position of cuckoos for which new egg is to be generated: CS
- list of transitions covered: Tlist
- Offset for the next state to be visited from present state: stepsize

The algorithm for this technique is given in Listing 1.

Each state in the input state diagram is considered as a nest. Each nest is a six tuple consisting of egg, number of outbounds, remaining outbounds, frequency, criticality and distance from final node as the attributes.

Each nest is a six tuple node with following elements:

$$\{\text{Egg}, \text{OB}, \text{criticality}, \text{frequency}, d_{\text{final}}, \text{OB}_{\text{remaining}}\}$$

Sections 4.1 and 4.2 describe the functions levyFlight() and objectiveFunction() used in the algorithm.

4.1. Levy Flight Function

This function is used to generate the value which tells about the next state to be traversed. In the implementation of the algorithm, the Levy flight function can be calculated using the McCulloch's algorithm (Leccardi, 2005).

The Gaussian case has been used in the algorithm, whose formula is given by:

$$Levy \sim u = t^{-\lambda},\ (1 < \lambda \le 3) \tag{1}$$

where c is scale factor & τ is location parameter.

The scale factor can be kept one, as each step in traversing the state diagram is of unit length. Furthermore, τ value in this particular case is 0 as the present state can be considered as the present position of cuckoo. The value of w and φ used in the equation (1) as given by McCulloh (Leccardi, 2005) is:

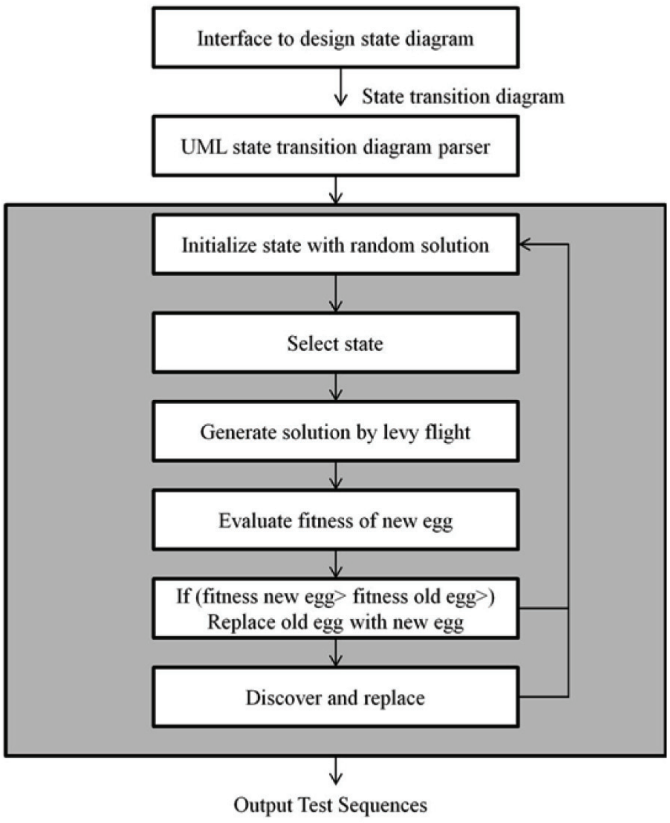
$$\text{w} = -\log_e(\text{random}()) \tag{2}$$

$$\varphi = (random() - 0.5) \times \pi \tag{3}$$

Here, the function random() gives a random value between 0 and 1.

The fractional part of 'x' is dropped and integer value is used for further calculation. If this value is greater than OB for present state, x = OB is used and if the value is less than 1, x = 1 is used.

Figure 1. Architecture of cuckoo search based algorithm



Interface to design state diagram
State transition diagram
UML state transition diagram parser
Initialize state with random solution
Select state
Generate solution by levy flight
Evaluate fitness of new egg
If (fitness new egg> fitness old egg>) Replace old egg with new egg
Discover and replace
Output Test Sequences

All the children of the present state are organised in an array in the sequence of the unique names or number assigned to states. The levyFlight() function returns the x^{th} child in the array of the present node as output.

The fitness of solution returned by levyFlight() is calculated by the objective function described in Section 4.2.

4.2. Objective Function

Objective function calculates the fitness of the new solution. The fitness function basically takes into account four factors namely frequency, criticality, distance of the state from initial node and final node. Logically, it is desirable that fitness must increase with criticality and frequency of the state. Also as longer test sequences are preferred, distance from final node must be directly proportional. Keeping these in consideration following formula is used for the objective function:

$$\left(\text{criticality} + \text{frequency}\right) \times (d_{final} + 1) \tag{4}$$

For example, if the value of the criticality and frequency are 1 and 3 respectively and the distance from final state is 2, then the value of objective function is calculated as:

$$\text{objectiveFunction}\left(\right) = \left(1 + 3\right) \times (2 + 1)$$

$$\text{==>} \text{objectiveFunction}\left(\right) = 12$$

Listing 1.

```
BEGIN.
Step 1. Each node's old egg is given a randomly chosen state out of its out-
bounds.
Step 2. Push start state in CS.
Step 3. While(CS! = empty){
    Step 3.1.  Get a cuckoo
        Cuckoo=Pop from CS

    The pop function used is standard pop( ) function of stack. This
    removes and returns the top value from CS stack (Hubbard 2000).
    Step 3.2. Generate its solution by performing Levy flights
        newEgg =  levyFlight( )

    levyFlight( ) function generates the value which tells about the next
    state to be traversed. The levyFlight( ) function is discussed in
    section 4.1.
    Step 3.3. Evaluate fitness value of newEgg (F1)
        F1 = objectiveFunction( )

    Objective function calculates the fitness of a solution. The objective
    function is discussed in section 4.2.
    Step 3.4 Evaluate the fitness of old solution (F2)
        F2 = objectiveFunction( )
    Step 3.5 Replace egg of present node if the new egg is better
        If(F1 > F2)
        Egg = newEgg
    Step 3.6. Worst nests are discovered and abandoned with probability Pa and
    new ones are built.
        Step 3.6.1 Discover eggs with probability Pa
                Generate α = random value
                between 0 and 1
                Discovered if α > Pa

                The value of Pa was kept at 0.25 as suggested by Yang and
                Deb(2009).
        Step 3.6.2 If the nest is discovered then produce biased solution
                The biased random walk is achieved by applying the following
                calculations
                        stepSize = random() × nest(randomPermutaion(n),:)-
                        nest(randomPermutation(n),:));

                        newNest = nest+stepsize×K;

                K = random(size(nest)) and K>pa, where pa is the discovery
                rate. Here random() is a random number generator between 0 and
                1, randomPermutation() returns a row vector of random
                permutation of integers between 1 and n. This is derived from a
                standard Cuckoo Search MATLAB implementation by Yang(2010).

    Step 3.7  Keep the best solutions/nests:
        If the transition from cuckoo to new egg is already traversed then,
                Discard newEgg.
                Push cuckoo in CS.
                Continue the next iteration in the while loop.
        Else
                Add the transition from cuckoo to new egg in TList.
                Decrease the remaining OB of present state by 1.
    Step 3.8. If any outbound of cuckoo are remaining from being traversed i.e.
```

continued on the following page

Listing 1. Continued

```
    OB remaining  ! = 0  then,
        Push cuckoo in CS.
    Step 3.9. If the OB of egg is not 0 then
        Consider Egg as a new cuckoo bird and Push in CS.
    Step 3.10. End while

Step 4. Print all the sequences in TList

END.
```

If any self loop or a transition to a state which is directly connected to final state is possible from the present node, the transition is given infinite value and the above function is not used.

If the transition is already traversed then a value (-1) is returned.

The algorithm described in preceding sections is explained with an example in the next section.

5. EXPERIMENTAL RESEARCH

The proposed algorithm strives for exhaustive coverage with minimal repetition which ensures all transitions coverage and all paths coverage at least once with minimal number of repetitions of states as well as transitions. The algorithm works by maximising an objective function which focuses on most error prone parts of the program so that critical portions can be tested first. The algorithm has been applied on various real life and like ATM banking system, ping pong video game software, enrolment system and few open source software and many more. A case study has been illustrated here to show the implementation of the algorithm.

The Figure 2 explains the state chart diagram of an Enrolment system which was used by (Doungsa-ard, 2005). An enrolment system diagram describes the activity of the enrolment for each course. The students enroll for the course. When the course is full, no more students can enroll for the course. The course can be closed for enrolment anytime.

To reduce the complexity of referencing the states, the states of the Figure 2 for further discussion are abbreviated as: Initial = S_1, Proposed = S_2, Scheduled = S_3, open for enrolment = S_4, Full = S_5, Closed to enrolment = S_6, Final state = S_7.

The input values provided by tester for the simulation are listed in the Table 1. The values for criticality and frequency are based upon the requirement analysis of the software. Criticality is the relative measure of difficulty in coding the functionalities assigned to a state. Here the criticality is assigned on a scale of 1 to 5. The result of the proposed algorithm is independent of the scale chosen. Frequency is the number of times a particular state is visited while completing the functionality of all the use cases.

Initially cuckoo is at start state i.e., S_1. The value of the randomly selected old egg is S_2.

The levyFlight() function calculates the value of 'x' as follows:

$w = -\log_e(\text{random}())$ (from eq. (2))
==>w= 0.3628810

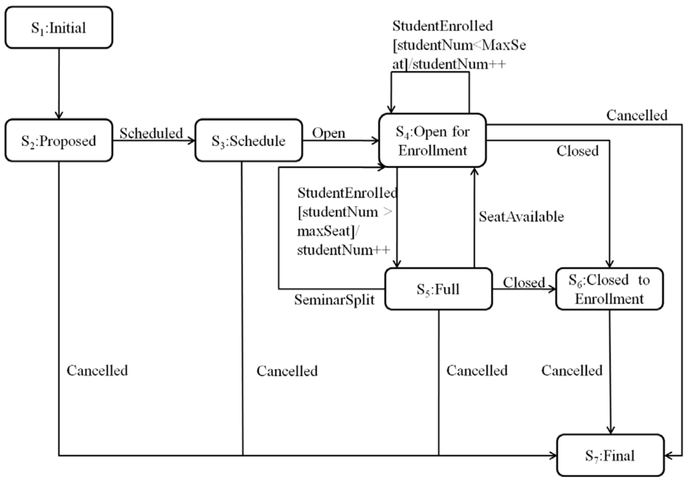
$\varphi = (random() - 0.5) \times \pi$ (from eq. (3))
==>φ =1.084979

Now, $x = c \times 2\sqrt{w} \times \sin\left(\varphi\right) + \tau$
(from eq. (1))

Here c = 1 and τ = 0 as discussed earlier in Section 4.1. Therefore the value of x is calculated as

$x = 2\sqrt{}(0.3628810)\sin(1.084979)$
==>x = 1.065

Figure 2. State diagram of enrolment system



The array of children of node S_1 is $\{S_2\}$. Therefore, value returned by levyFlight() is first child of present node i.e., S_2.

Now, the fitness of new egg is calculated. The values of criticality and frequency as input from user are 1 and 7 respectively. The shortest distance of S_2 from final state is 1. Therefore,

$$F1 = (1+7)\times(1+1) \text{ (from eq. (4))}$$
$$==>F1 = 16.$$

The old egg present at current nest is S_2, its fitness is calculated by objective function. This value is 16, as shown in Table 2. As the value of new egg is not greater than old egg, the egg need not be replaced.

The value of α generated in step 3.6.1 is 0.059 which is less than Pa (0.25). Hence new egg is not discovered by host bird. The transition $S_1 \rightarrow S_2$ is added to the list of transitions. The outbound of the state S_1 is decremented. The outbound of S_1 is now zero. So, it is not added to the stack. The state S_2 has an outbound of 2, therefore it is added to the stack.

In the next iteration, the cuckoo moves to S_2. The array of children of node S_2 is $\{S_3, S_7\}$. The value of x, generated in levyFlight() is 2.238, only the integer part of this value is taken thus the second child of S_2 i.e., S_7 is selected.

Now the fitness of new egg is calculated. According to the algorithm of objective function, the fitness value is infinite. Instead of

Table 1. Input data for simulation

State	Criticality	Frequency
S_1	1	7
S_2	1	7
S_3	1	6
S_4	5	5
S_5	4	4
S_6	2	2
S_7	1	7

Table 2. Data table for each iteration

Iteration	Node	Old Egg	x(from eq. (1))	newEgg	Fitness old egg	Fitness newEgg	α (in step 3.6.1)	Egg discovered (α<Pa)	Transition covered	Stack
1	S_1	S_2	1.065	S_2	16	16	0.059	False	$S_1 \rightarrow S_2$	S_2
2	S_2	S_3	2.238	S_7	14	2147483646	0.099	False	$S_2 \rightarrow S_7$	S_2
3	S_2	S_7	-0.612	S_3	-1	14	0.189	False	$S_2 \rightarrow S_3$	S_3
4	S_3	S_4	2.683	S_7	20	2147483646	0.067	False	$S_3 \rightarrow S_7$	S_3
5	S_3	S_7	0.819	S_4	-1	2147483646	0.523	True	$S_3 \rightarrow S_4$	S_4
6	S_4	S_4	1.065	S_4	2147483646	2147483646	0.009	False	$S_4 \rightarrow S_4$	S_4
7	S_4	S_4	2.686	S_5	-1	2147483646	0.313	True	$S_4 \rightarrow S_7$	S_4
8	S_4	S_7	2.417	S_5	-1	2147483646	0.145	False	$S_4 \rightarrow S_5$	S_4,S_5
9	S_5	S_4	5.087	S_7	20	2147483646	0.019	False	$S_5 \rightarrow S_7$	S_4,S_5
10	S_5	S_7	3.428	S_6	-1	2147483646	0.149	False	$S_5 \rightarrow S_6$	S_4,S_5,S_6
11	S_6	S_7	1.065	S_7	2147483646	2147483646	0.171	False	$S_6 \rightarrow S_7$	S_4,S_5
12	S_5	S_6	0.892	S_4	-1	20	0.004	False	$S_5 \rightarrow S_4$	S_5,S_4
13	S_4	S_5	-0.298	S_6	-1	2147483646	0.219	False	$S_4 \rightarrow S_6$	S_5
14	S_5	S_4	2.256	S_4	20	20	0.044	False	$S_5 \rightarrow S_4$	empty

infinite, the implementation uses a very high integer value, which is (maximum value of integer in java)-1. The fitness of new egg F1 is 2147483646.

Fitness value of old egg is calculated according to the objective function formula (4). The value of old egg F2 is 14.

Now the value F1 and F2 is compared, the fitness value of new egg is greater. Therefore, S_7 replaces the old solution. In step 3.6 of algorithm the host bird is unable to detect the alien egg.

The transition $S_2 \rightarrow S_7$ is added to the list of transitions. The state outbound of the state S_2 is decremented. The outbound of S_2 is still greater than zero i.e. 1. It is added again to the stack. The state S_7 has 0 outbound, therefore it is not added to the stack. In this way the test path generated is $S_1 \rightarrow S_2 \rightarrow S_7$.

In this sequence the transitions generated are trivial. Another intricacy may arise when the alien egg is discovered in step 3.6 of the proposed algorithm and destroyed subsequently. For illustration consider iteration 5 in above table. Here present nest of cuckoo is S_3 and old egg value is S_7. The array of children for state S_3 is {S_4, S_7}.

The value of x, generated in levyFlight() function is 0.819, which is less than 1. According to levyFunction() in Section 4.1, the value returned is 1 and hence the newEgg is S_4.

The values of F1 and F2 are calculated as 2147483646 and -1 respectively. As F1 > F2, newEgg replaces the old egg in the nest.

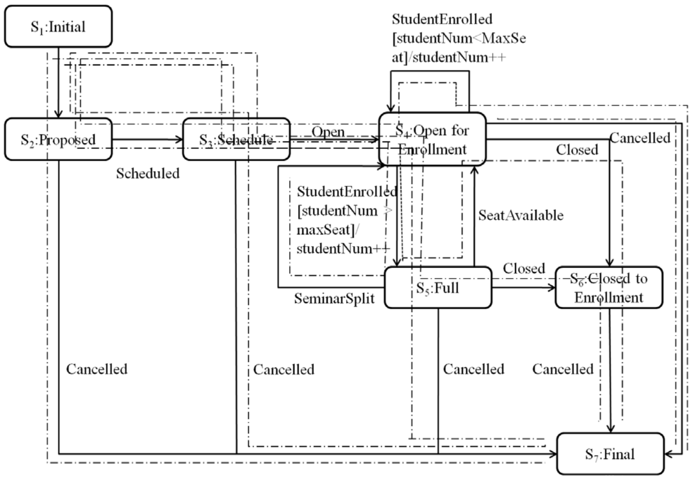
Now, in step 3.6.1 value of α generated is 0.523. Here α > Pa, which means the newEgg in the nest is discovered by the host bird and is destroyed. In step 3.6.2, a biased solution is produced to replace the egg which is destroyed by the host bird. The new solution requires following calculations:

$$\text{random}\left(\right) = 0.723$$
$$\text{stepsize} = 0.723 \times 1 = 0.723$$

Here stepsize is rounded to an integer as stepsize of the Cuckoo bird must be an integer. Therefore, stepsize is taken as 1.

$$\text{newEgg} = 0.723 \times 1 + \left(1 - 0.723\right) \times 1$$
$$==> \text{newEgg} = 1.723$$

Figure 3. State diagram showing generated test sequences



Again, the fractional part of the newEgg value is truncated to get the final value of newEgg as 1. Now, the array of children for state S_3 is $\{S_4, S_7\}$, whose first child is returned as the new biased solution and the transition $S_3 \rightarrow S_4$ is traversed.

In similar way, the whole state diagram is traversed until all the transitions are covered and the sequences generated are: $S_1 \rightarrow S_2 \rightarrow S_7$, $S_2 \rightarrow S_3 \rightarrow S_7$, $S_3 \rightarrow S_4 \rightarrow S_4 \rightarrow S_7$, $S_4 \rightarrow S_5 \rightarrow S_7$ $S_5 \rightarrow S_6 \rightarrow S_7$, $S_5 \rightarrow S_4 \rightarrow S_6$ and $S_5 \rightarrow S_4$.

These sequences can be concatenated to produce minimal sequences: $S_1 \rightarrow S_2 \rightarrow S_7$, $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_7$, $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_4 \rightarrow S_7$, $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_5 \rightarrow S_7$, $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_5 \rightarrow S_6 \rightarrow S_7$, $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_5 \rightarrow S_4 \rightarrow S_6 \rightarrow S_7$ and $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_5 \rightarrow S_4 \rightarrow S_7$. The sequences generated through the algorithm are shown in the Figure 3.

The approach is able to provide test sequences with full coverage. The sequences generated are not fully optimal but the optimality has been achieved to a large extent.

6. ANALYSIS

In the current section the results of proposed algorithm have been analysed and compared statistically with two existing techniques, genetic algorithm (Doungsa-ard et al., 2005) and ant colony optimisation (Srivastava & Baby, 2010). The approaches (Doungsa-ard et al., 2005; Srivastava & Baby, 2010) have contributed significantly and provided phenomenal results by using meta-heuristic approaches in the field of software testing. The present algorithm strives to continue the research further in this field and provide even better results.

Genetic algorithm based approach of test sequence generation is used in paper (Doungsa-ard et al., 2005). The genetic algorithm is applied by creating objective function for each path. GAs also generates subpopulation for each objective function individually. The transitions which are fired and contained in the possible paths are counted as a fitness value. The overall coverage transition is calculated by selecting the best solution from each sub individual, and then executing the state machine diagram.

The ant colony optimization approach (Srivastava & Baby, 2010) can be used for test sequence generation using state machine diagram. Selection of the transition depends upon the probability of the transition. The value of transition depends upon direct connection between the vertices, heuristic information of the transition and the visibility of a transition for an ant at the current vertex.

Table 3. Table for comparative results of simulation

Algorithm	**Case Study**					
	Class management system		**Enrollment system**		**Telephone system**	
	Complete Coverage	Repetition Ratio	Complete Coverage	Repetition Ratio	Complete Coverage	Repetition Ratio
Cuckoo Search	Yes	1.67	Yes	3	Yes	2.11
Genetic Algorithm	No	2.33	No	4.43	No	2.89
Ant Colony Optimization	Yes	3.00	Yes	5.43	Yes	3.11

The genetic algorithm based approach requires to set many parameters like crossover probability, mutation probability and population size etc. Similarly, in ant colony optimization approach probability of selecting a transition depends upon pheromone value, feasibility of transition, heuristic information of transition, desirability and visibility and other parameters.

For optimal performance of the algorithm (genetic or ant), setting these parameters to correct values is a crucial task. But finding the correct value for these parameters requires elaborate series of experiments, which turns out to be a tedious task. Whereas, present cuckoo based approach requires setting only three parameters, criticality & frequency of a state in state diagram and probability of discovery by host bird (Pa).

Proposed algorithm uses the Cuckoo Search based approach to traverse the state diagram. Using the objective function based on four factors namely frequency, criticality, distance of the state from initial node and final node, makes the algorithm more efficient. Any weak solution generated in the procedure is discovered by the host bird and replaced by generating a biased solution.

The results in previous section affirm that the proposed algorithm is efficient. The results of the proposed approach are compared with results of genetic algorithm and ant colony optimization. Comparative results of simulation on several real life examples from (Doungsa-ard et al., 2005) are shown in Table 3.

Also, statistical data about comparison between proposed algorithm, genetic algorithm and ant colony optimization is plotted in Figure 4 and Figure 5.

For each of the three algorithms, the experiments were carried out in conditions to deliver best results

The parameters for genetic algorithm based approach, as suggested by the approach (Doungsa-ard et al., 2005), are set as follow:

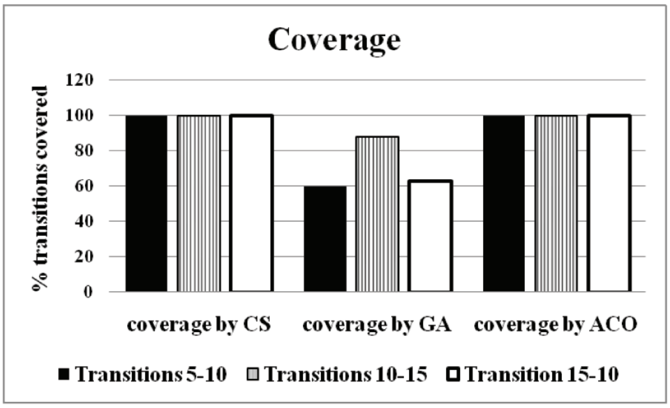
- Crossover probability = 0.5.
- Mutation probability = 0.05.
- Size of population in each generation for each objective function = 10.

The chromosome length which covers maximum number of transitions is used for calculations. For simulation of ant colony approach, the parameters heuristic value (η) and pheromone level (τ) were set with the values suggested in Srivastava and Baby (2010) as 2 and 1 respectively. For Cuckoo Search based approach, we have kept the probability of discovery (Pa) at 0.25, as suggested by Yang and Deb (2009).

Each experiment was run 20 times to eliminate the possibility of stochastic results. The average over these 20 observations was used for calculation of coverage and nodal visits.

The two algorithms (Doungsa-ard et al., 2005; Srivastava & Baby, 2010) along with present approach were simulated on a number

Figure 4. Comparison of coverage



of real life as well as theoretical problems. The comparative study depicts the relative strengths of these algorithms. The genetic algorithm based approach tries to cover the critical portions of the state diagram, but it fails to provide full coverage in all three problems. The ant colony optimisation approach provides 100 percent coverage of transitions in the state diagram, though the ratio of the number of transitions covered to the number of transitions present in the state diagram is comparatively poor. The present Cuckoo Search based approach provides better repetition ratios along with complete coverage in each of the case studies.

Figure 4 shows the comparison of percentage coverage of these approaches. As visible from the graph, Cuckoo search and ant colony based approach shows much better coverage than genetic algorithm based approach.

Figure 5 shows the graph between the number of nodes traversed in the test sequences. This graph shows the variation of traversed nodes with the total number of nodes present in the graph. On the basis of the general observations in the graph, it can be concluded that results produced by Cuckoo search algorithm are free from any convergence towards local optima to a large extent.

Figure 5. Plot of nodal visits of proposed algorithm

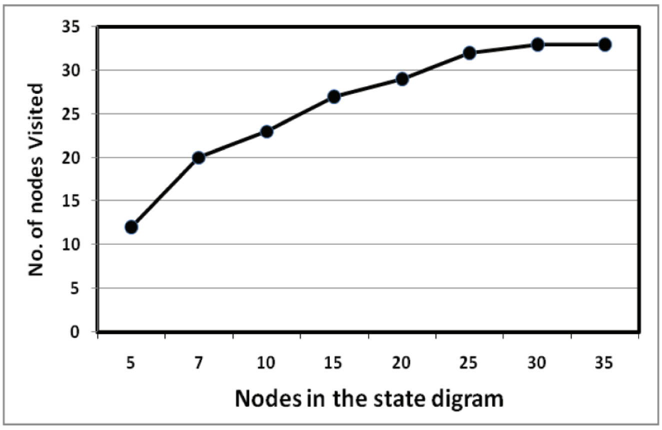


Figure 6. Comparison of optimality

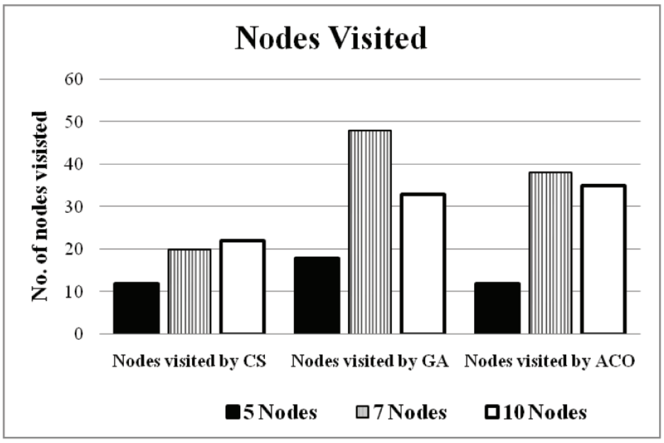


Figure 6 shows the comparison of the number of the nodes visited by the three approaches when applied to same state diagrams. The optimal transitions are obtained when all transitions can be covered with least number of states visited. The Cuckoo search based approach shows better results than both genetic algorithm and ant colony optimisation based approach.

7. CONCLUSION

This paper presented a Cuckoo search optimization approach for test sequence generation on state transition diagram. A UML diagram is created to represent the State chart model of a software system under test. Using the developed Cuckoo search based meta-heuristic algorithm, optimal sequences of transitions in state model can be generated to achieve test coverage requirement.

Statistical results confirmed that the present approach as compared to previously employed meta-heuristic approaches of test sequence generation like genetic algorithm (Doungsa-ard et al., 2005) and Ant colony optimisation (Srivasatava & Baby, 2010), performs better in terms of coverage and optimality of test sequences.

REFERENCES

Alander, J. T., Mantere, T., & Turunen, P. (1998). Genetic algorithm based software testing. In *Proceedings of the 3rd International Conference on Artificial Neural Networks & Genetic Algorithms*, Wein, Austria (pp. 325-328).

Blaha, M. R., & Rumbaugh, J. R. (2011). *Object oriented modeling and design with UML* (2nd ed.). Upper Saddle River, NJ: Pearson.

Chow, T. (1978). Testing software designs modeled by finite-state machines. *IEEE Transactions on Software Engineering*, *4*(3), 178–187. doi:10.1109/TSE.1978.231496

Dijkstra, E. W. (1970). *Notes on structured programming* (2nd ed.). Eindhoven, The Netherlands: Technological University Eindhoven Academic Press.

Doungsa-ard, C., Dahal, K., Hossain, A., & Taratip, S. (2005). An improved automatic test data generation from UML state machine diagram. In *Proceedings of the Fifth International Conference on Quality Software*, Melbourne, Australia (pp. 255-264).

Edgar, R. (1989). Structured programming. *Dr. Dobb's Journal of Software Tools for the Professional Programmer*, 56-60.

Fujiwara, S., Bochman, G., Khendek, F., Amalou, F., & Ghedasmi, A. (1991). Test selection based on finite state models. *IEEE Transactions on Software Engineering*, *17*(6), 591–603. doi:10.1109/32.87284

Gandomi, A. H., Yang, X. S., & Alavi, A. H. (2011). Cuckoo Search Algorithm: A metaheuristic approach to solve structural optimization problems. *Engineering with Computers*, 27.

Hubbard, J. R. (2000). *Data structures with C*. New York, NY: McGraw-Hill.

Kaner, C. (2006). Exploratory testing. In *Proceedings of the Quality Assurance Institute Worldwide Annual Software Testing Conference*, Orlando, FL.

Leccardi, M. (2005). Comparison of three algorithms for Lèvy Noise Generation. In *Proceedings of the Fifth EUROMECH Nonlinear Dynamics Conference* (pp. 1-6).

Li, H., & Lam, C. P. (2004). Optimization of state-based test suites for software systems: An evolutionary approach. *International Journal of Computer & Information Science*, *5*(3), 212–223.

Li, K., Zhang, Z., & Liu, W. (2009). Automatic test data generation based on ant colony optimization. In *Proceedings of the Fifth International Conference on Natural Computation* (pp. 216-219).

Mayers, J. G. (2004). *The art of software testing* (2nd ed.). Hoboken, NJ: John Wiley & Sons.

Naito, S., & Tsunoyama, M. (1981). Fault detection for sequential machines by transition tours. In *Proceedings of the Conference on Fault Tolerant Computing Systems*, Stanford, CA (pp. 238-243).

Offutt, A. J., Liu, S., Abdurazik, A., & Ammann, P. (2003). Generating test data from state-based specifications. *The Journal of Software Testing, Verification & Reliability*, *13*(1), 25–53. doi:10.1002/stvr.264

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, *15*(5), 330–336. doi:10.1145/355602.361309

Payne, R. B., Sorenson, M. D., & Klitz, K. (2005). *The Cuckoos*. New York, NY: Oxford University Press.

Pressman, R. (2001). *Software engineering–A practitioner's approach* (5th ed.). New York, NY: McGraw-Hill.

Rathore, A., Bohara, A., Gupta, P. R., Lakshmiprashanth, T. S., & Srivastava, P. R. (2011). Application of genetic algorithm and tabu search in software testing. In *Proceedings of the Fourth Annual ACM Bangalore Conference*.

Sokenou, D. (2006). Generating test sequences from UML sequence diagrams and state diagrams. *Informatik f ̈ur Menschen, 2*(94), 236-240.

Speed, E. R. (2010). Evolving a Mario agent using Cuckoo Search and Softmax Heuristics. In *Proceedings of the Games Innovations Conference*, Hong Kong (pp. 1-7).

Srivastava, P. R., Jose, N., Barade, S., & Ghosh, D. (2010a). Optimized test sequence generation from usage models using ant colony optimization. *International Journal of Software Engineering and Applications*, *1*(2), 14–28. doi:10.5121/ijsea.2010.1202

Srivastava, P. R., Jain, P., Baheti, S., & Samdani, P. (2010b). Test case generation using genetic algorithm. *International Journal Information Analysis and Processing*, *3*(1), 7–13.

Srivastava, P. R., Ray, M., & Raghurama, G. (2009). Selection of automated functional and regression testing tool using analytic hierarchy process (AHP) method. *ACM SIGSOFT Software Engineering Notes*, *34*(2), 1–4. doi:10.1145/1507195.1507216

Turner, C. D., & Robson, D. J. (1993). The state-based testing of object-oriented programs. In *Proceedings of the IEEE Conference on Software Maintenance*, Montreal, QC, Canada (pp. 302-310).

Yang, X.-S. (2010). *Cuckoo Search (CS) algorithm - file exchange - MATLAB Central*. Retrieved September 5, 2011, from p://www.mathworks.com/matlabcentral/fileexchange/29809-cuckoo-search-cs-algorithm

Yang, X. S., & Deb, S. (2009). Cuckoo Search via Lévy Flights. In *Proceedings of the World Congress on Nature & Biologically Inspired Computing* (pp. 210-214).

Praveen Ranjan Srivastava is working with the Software Engineering and Testing Research Group in the Computer Science and Information Systems Department at the Birla Institute of Technology and Science (BITS) Pilani, Pilani Campus Rajasthan, India. He is currently doing research in the area of software testing. His research areas are software testing, quality assurance, quality attributes ranking, testing effort, software release, test data generation, Test effort estimation, agent oriented software testing, and metaheuristic approaches. He has published more than 70 research papers in various leading international journals and conferences in the area of software engineering and testing. He has been actively involved in reviewing various research papers submitted in his field to different leading journals and various international and national level conferences. He received various funds from different agencies like Microsoft, IBM, Google, DST, CSIR, etc.

Ashish Kumar Singh is pursuing Master of Engineering specializing in Software Systems from Computer Science and Information Systems Department at the Birla Institute of Technology and Science (BITS) Pilani, Pilani Campus Rajasthan, India.

Hemraj Kumhar is pursuing Master of Engineering specializing in Software Systems from Computer Science and Information Systems Department at the Birla Institute of Technology and Science (BITS) Pilani, Pilani Campus Rajasthan, India.

Mohit Jain is pursuing Master of Engineering specializing in Software Systems from Computer Science and Information Systems Department at the Birla Institute of Technology and Science (BITS) Pilani, Pilani Campus Rajasthan, India.