ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/277714470

Favourable test sequence generation in state-based testing using bat algorithm

Article *in* International Journal of Computer Applications in Technology · July 2015

DOI: 10.1504/IJCAT.2015.070495

CITATION
1

READS
76

1 author:

Dr. Praveen Ranjan Srivastava
Indian Institute of Management, Rohtak
113 PUBLICATIONS **844** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Management Information Systems View project

Big data Analytics View project

All content following this page was uploaded by Dr. Praveen Ranjan Srivastava on 26 November 2015.

The user has requested enhancement of the downloaded file.

Favourable test sequence generation in state-based testing using bat algorithm

Praveen Ranjan Srivastava*

Information Technology & Systems Group,
Indian Institute of Management (IIM),
Rohtak 124001, India
Email: praveenrsrivastava@gmail.com
*Corresponding author

Kumar Pradyot, Deepshikha Sharma and K.P. Gouthami

Department of Computer Science,
Birla Institute of Technology and Science (BITS),
Pilani, Rajasthan 333031, India
Email: pradyot21@gmail.com
Email: deepshikhas17@gmail.com
Email: gautami.kp@gmail.com

Abstract: Software testing has always been a significant component of the software development life cycle. Also, in any project a good amount of time and cost is required for testing. For optimal results, maximum testing in minimum time is desired with fewest repetitions. In large and complex systems, this is possible only through use of meta-heuristic algorithms to help decide which portions to test first. A similar approach is presented in this paper which demonstrates how the bat algorithm works upon a given state chart diagram and suggests favourable test sequences keeping in mind the critical system modules, which often need to be tested first. A graphical representation and comparative results with other algorithms currently in use reflect the basis of choosing the new alternative technique that this paper presents.

Keywords: state-based testing; bat algorithm; meta-heuristics; nature inspired algorithm; software testing; test sequence generation.

Reference to this paper should be made as follows: Srivastava, P.R., Pradyot, K., Sharma, D. and Gouthami, K.P. (2015) 'Favourable test sequence generation in state-based testing using bat algorithm', *Int. J. Computer Applications in Technology*, Vol. 51, No. 4, pp.334–343.

Biographical notes: Praveen Ranjan Srivastava is working as an Assistant Professor in the Information Technology and Systems Group at Indian Institute of Management (IIM), Rohtak, India. He is currently doing research in the area of Software Engineering and Management using novel meta-heuristic techniques. His research areas are software testing, quality and effort management; software release management, test data generation, decision science and business analytics. He has published more than 120 research papers in various leading international journals and conferences in the area of research. He is Editor-in-Chief of *International Journal of Software Engineering Application and Technology (IJSEAT)*, published by Inderscience. He is also member of Editorial Board of various leading journals.

Kumar Pradyot is pursuing Master of Engineering specialising in Software Systems from Computer Science and Information Systems Department at the Birla Institute of Technology and Science (BITS) Pilani, Pilani Campus Rajasthan, India. His interests lie in software quality assurance, operating systems, computer networks and cloud computing.

Deepshikha Sharma is pursuing Master of Engineering specialising in Software Systems from Computer Science and Information Systems Department at the Birla Institute of Technology and Science (BITS) Pilani, Pilani Campus Rajasthan, India. Her interests lie in software quality assurance, operating systems, cloud computing and object oriented design.

K.P. Gouthami is pursuing Master of Engineering specialising in Software Systems from Computer Science and Information Systems Department at the Birla Institute of Technology and Science (BITS) Pilani, Pilani Campus Rajasthan, India. Her interests lie in software testing, distributed operating systems, object oriented design and databases.

Copyright © 2015 Inderscience Enterprises Ltd.

1 Introduction and background

The continuously increasing visibility of software services and their associated failure costs motivate the software researchers to go for well-planned and thorough testing (Pressman, 2001). Software testing phase has become a very important part. With the need for highly reliable, scalable and robust software, it is seen throughout the software engineering paradigm that testing accounts for almost 50% of the total effort. Thus, the automation of software testing is very essential (Pressman, 2001). Also, in extreme cases, testing of human-rated software (e.g., flight control, nuclear reactor monitoring) can cost three to five times as much as all other software engineering steps combined.

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and code generation (Pressman, 2001).Testing software thoroughly ensures that the customer satisfaction is met and increases the reliability of the program under test. Software testing is a very broad sub process in the software development life cycle, which a potpourri of methods existing to test different aspects of a piece of code. Black-box testing (Pressman, 2001), as the name suggests, tests a program for its functional accuracy and precision, without peeking at the internals. On the other hand, White-box testing (Pressman, 2001) evaluates and validates on the very basis of the program structure including in-depth analysis of line by line code. They cover almost all of the testing techniques currently in use. Other testing methods have come up recently due to changes in software technology, such as for testing a user interface etc. (Pressman, 2001).

Nowadays, it is common to come across software sizes running into thousands and millions of lines of code, and such large systems cannot be practically tested using manual or random testing and automated testing tools are required to reduce test effort and improve testing efficacy (Mayers, 2004).

It is known (Pressman, 2001) that not all possible test cases can be tested in polynomial time owing to the problem showing characteristics typical of NP Hard complex problems (Alander et al., 1998). This amalgamates artificial intelligence concepts into software testing and discards the traditional approach to give way to much faster techniques based on heuristics and meta-heuristics (Srivastava et al., 2010a, 2010b; Doungsa-ard et al., 2005). These terms often describe the primary subfield of stochastic optimisation.

A research based on ant colony performed by Li and Lam (2004), produced sequences which were usually much longer than the shortest possible sequences, nevertheless provided complete coverage.

An approach to obtain test sequence-based coverage of state machine diagram using genetic algorithms by Doungsa-ard et al. (2005) establishes the criteria of the quality of the test sequences. It examines the test cases for criticality and other factors, yet the proposed approach failed to provide complete coverage.

Work done by Li and Lam (2004) and Srivastava et al. (2010a), to generate test data based on ant colony optimisation technique along with the branch function technique and problem of local optimisation has also been solved. The model takes into account factors such as cost and criticality of various states in the model. In these papers, although the most critical nodes have been covered, complete state and transitions coverage has not been achieved.

In 2011, an approach combining genetic algorithm along with Tabu search by Rathore et al. (2011), uses control-dependence graph. The objective function used in the approach fails to consider a number of factors such as distance from initial and final node, criticality, etc.

The year 2012 saw two techniques namely the Cuckoo Search (Gandomi et al., 2011; Srivastava et al., 2012a) and Intelligent Water Drop Algorithm (Srivastava et al., 2012b), which also attempted to minimise repetitions and give complete coverage. Although both the algorithms promise complete traversal of the test domain, they lack in optimising the test sequences generated by the respective approaches.

In spite of a lot of work done for finding optimal or favourable test sequences, efforts have resulted in very little success. The proposed approach here tries to resolve the problem towards exhaustive state coverage and optimal sequence generation.

This paper is structured as follows. Section 2 discusses the related terminology. Section 3 describes the bat algorithm and other techniques used in the proposed approach. Section 4 presents the details of the proposed approach for test sequence generation. Section 5 illustrates the research work by including a case study. Section 6 compares the presented approach with other existing techniques for test sequence generation, Section 7 discusses drawback and future work and finally Section 8 concludes the paper.

2 Related concepts

This section lists and explains the prerequisite terminology and briefs the reader about some of the similar techniques introduced in other papers. To understand the basics of the bat algorithm and how it has been modified to fit over the testing platform, we must understand the following.

2.1 State-based testing (Blaha and Rumbaugh, 2011)

A state diagram is a graph whose nodes are states and directed arcs are transitions between states. It specifies the possible states and what transitions are allowed between states, what events cause the transitions to occur and what behaviour is executed in response to events. With respect to software systems, a state diagram represents the states a particular software may possibly be in, and the events which may lead to a change in its state. Essentially, it covers the scope of any software system. State-based testing is a part

of black box testing wherein the goal of the tester is to traverse the entire state diagram while reducing the number of repetitions in the sequences.

2.2 Meta-heuristics (Luke, 2012; Yang and Deb, 2009; Srivastava, 2010)

Meta-heuristics comes under stochastic optimisation, which is the general class of algorithms which employ some degree of randomness to find optimal (or as optimal as possible) solutions to hard problems. They are the most generalised of the algorithms and can thus be applied to a wide range of problems. They are used to find answers to problems when brute force techniques fail due a large search space and we have little idea what the solution might look like or how to proceed in a principled way.

2.3 Nature inspired approaches (Yang, 2010)

About a trillion species of living beings inhabit the earth. Each species has its own way to live, respire, migrate and reproduce. This gives scientists a lot to study and learn from and use this information elsewhere. The behaviour of biological systems and/or physical systems in nature has been the source of derivation of several heuristic and meta-heuristic algorithms. Genetic algorithm, ant colony optimisation and cuckoo search are all nature inspired algorithms to name a few.

2.4 Objective function and random walk (Yang, 2010)

These two are subcomponents of a meta-heuristic algorithm and help it narrow down on a solution, without which the algorithm cannot converge. Fitness function is a type of objective function used to summarise how close a design solution is to achieving the set aims. The global best solution of a meta-heuristic algorithm depends on maximisation and minimisation of certain parameters. Random walk refers to the semi random movement of the species. It is basically a mathematical function dependent on at least one random variable.

This paper discusses a relatively new meta-heuristic technique known as the Bat Algorithm (Yang, 2010) to improve upon existing techniques in state-based testing.

3 The bat algorithm

This section throws light upon the behaviour of bats, the species from which the algorithm has been inspired from which is directly taken by following reference.

3.1 Behaviour of bats (Yang, 2010; Altringham, 1996)

Bats are fascinating animals (Yang, 2010). They are the only mammals with wings and also have advanced capability of echolocation. Most bats uses echolocation to a certain degree among which microbats are a popular example as they use echolocation extensively while megabats do not. This echolocation is a type of sonar to detect prey, avoid obstacles, and locate their roosting crevices in the dark. These bats emit a very loud sound pulse and listen for the echo that bounces back from the surrounding objects. These pulses vary in properties and depending on the species, change with their hunting strategies.

The loudness also varies from the loudest when searching for prey and decreases when homing towards the prey. The travelling range of such short pulses varies from a few metres, depending on the actual frequencies. Such echolocation behaviour of microbats can be defined in such a way that it can be associated with the objective function to be optimised, which makes it possible to formulate new optimisation algorithms.

3.2 The bat algorithm (Yang, 2010)

The bat algorithm was developed by Yang (2010). It is a nature inspired meta-heuristic algorithm ideal for optimising mathematical problems of NP hard complexity. He proposed that on idealising the echolocation characteristics of microbats described above, we can develop bat algorithms with a few assumptions which are (Yang, 2010):

- All bats use echolocation to sense distance, and they also 'know' the difference between food/prey and background barriers in some unknown way.
- Bats fly randomly with velocity v_i at position x_i with a fixed frequency f_i, and loudness A_0 to search for prey. They can automatically adjust the frequency of their emitted pulses depending on the proximity of their target.
- Even though the loudness can vary in many ways, we assume that the loudness varies from a large positive A_0 to a minimum constant value $A_{\min}$.

The bats move about in a fashion as if each transition represents a local solution to close in the gap between them and their prey. A final state in a given state diagram represents a prey. Diagrams having multiple final states mean multiple prey for the bats, giving bats even the freedom to choose that prey which can be most easily captured.

As it is an iterative process each bat moves about on its own and searches for prey. After each iteration, the paths followed by the bats are ranked according to a fitness function. The solutions generated are ordered and the optimum of them is selected. The sequence stops when a bat captures its prey, or as in our case, a final state is encountered.

According to the algorithm, bat movement is performed through a random walk, given by a function dependent upon the loudness of emitted waves.

The next section discusses the algorithm's working and explains how the parameters are chosen and how they are formulated in the fitness function.

4 Details of proposed approach

This paper suggests a method for the generation of favourable test sequences of a state diagram given as input in order of their optimality using the bat algorithm. A favourable test sequence is one which takes into consideration practical factors and maximises coverage (Srivastava et al., 2012a). The purpose of the proposed solution is to have complete coverage with minimal redundancy.

The algorithm focuses on coverage of all paths having most desirable to least desirable transitions. This is done by minimising the fitness function of various states reached by bat through random walk.

The state diagram fed as input is designed using the Papyrus plugin in Eclipse IDE. A UML (Blaha and Rumbaugh, 2011) file is generated automatically by the plugin that stores the information about all states and their corresponding transitions. This file is parsed using a SAX Parser (Janert, 2002) to populate an appropriate data structure such as an adjacency matrix or a graph, which is used by the algorithm to generate the test sequences.

Table 1 lists the parameters used in the approach presented here. This table shows the relation between the parameters and the properties of the state diagram.

Table 1 Bat algorithm parameters

Parameter	*Description*
Frequency	Multiplicity of end states brings change in frequency. More end states more number of frequencies required
Velocity	Increases with increasing proximity to solution
Loudness	High Loudness away from solution, lowers down as bat approaches solution
Location	The new location for each bat is generated locally using random walk

The algorithm utilises certain parameters out of which a few help compute the bat movements and others fulfil the purpose of finding a local optimum amongst the obtained neighbouring solutions.

Table 2 depicts the parameters of the state diagram mapped with its properties. These parameters were chosen because certain parameters belonging to the state diagram are also needed to bring about more accurate results.

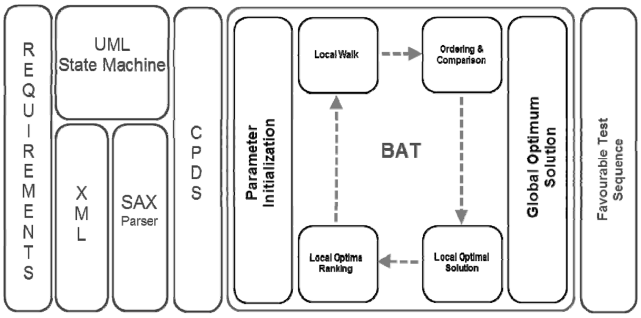
4.1 Architecture of algorithm

The proposed approach can be understood through a diagram depicting the technique's software architecture which has been presented in pipes and filters format. Each layer processes some input and passes on the output to the next layer. The implementation diagram for the solution is shown in Figure 1.

Table 2 State diagram parameters

Parameter	*Description*
Desirability	The criticality of each state ranging from 1 to 0 with 0 as the highest importance and 1 as least
Probability	Probability to reach end state. Depends on the number of final states

Figure 1 Architecture of the bat algorithm approach in software testing



The architecture contains four layers, namely

- software requirements
- UML representation
- programmable data structure representation
- bat algorithm core layer.
- *Software requirements*

This layer formulates the client requirements and specifications. This document will then be used in the graphical form of UML diagrams.

- *UML representation*

Many UML diagrams exist to represent client requirements to the project developer and tester. These diagrams reflect the actual work flow of any given software at that time. Each UML diagram is just another viewpoint of the software through different angles. State Chart Diagram is used in this project to find out the test sequences. As UML Diagrams cannot be read by the system directly, there is a need to convert it into a particular readable format. This is where the XML representation (Janert, 2002) comes in and each state diagram carries with itself a corresponding XML map.

- *Corresponding populated data structure (CPDS)*

The XML representation, though understandable by the system, is not suitable for logical operations. To perform programming and analyse this type of data, a SAX parser is used to populate a suitable data structure chosen by the testing team with the data input from the XML map. This is

a must before any logical operation or algorithm can be applied to the diagrammatic representation.

- *BAT algorithm core (Yang, 2010)*

The core layer of the project functions to generate all possible sequences and find the set of favourable sequences to be forwarded to the testing team. Certain associated parameters such as loudness, velocity, etc. are initialised and the random walk is performed by bats each time to obtain a local optimum solution. These solutions are then ranked on a fitness scale. This is an iterative process to get a global best solution.

Once the parameters are defined, the same are formulated into a fitness function to calculate optimality of a local solution.

4.2 Fitness function

Before the fitness can be determined, each parameter is monitored and certain calculations must be made. The equations involved in this process are:

Movement of bats from current position is updated by position x_i and velocity v_i at a time instant given by

$$f_i = f_{\min} + (f_{\max} - f_{\min})\beta, \tag{1}$$

$$v_i^t = v_i^{t-1} + (x_i^t - x_*)f_i, \tag{2}$$

where

- f_i is the frequency with which the bats find their prey. $f_{\min}$ and $f_{\max}$ are the minimum and maximum frequencies, respectively, which denote the range
- v_i is the velocity of the bat which increases as the bat reaches its prey.
- x^* is the local best location (solution) which is located after comparing the current solutions.
- β = [Desirability of final state * Probability to reach final state]($\beta \in [0, 1]$), Here, β is the factor that denotes dependency of paths chosen upon the importance of the end state the probability to reach there.

For the local search, once a solution is selected among the current best solutions, then by applying random walk, a new solution for each bat is generated locally (Yang, 2010).

$$x_{\text{new}} = x_{\text{old}} + € \times A_t, \tag{3}$$

where

- A_t is the loudness of the bat which decreases as the bat reaches its prey.
- € is a random integer chosen in the range [0,1]
- x_{old} is the local best solution
- x_{new} gives the random walk which varies for each bat.

The loudness A_i has to be updated accordingly as the iterations proceed. As the loudness usually decreases once a bat has found its prey (Yang, 2010), the loudness can be chosen as any value of convenience.

$$A_i^{t+1} = \alpha \times A_i^t \tag{4}$$

where α is a constant.

The fitness function has hence been selected as

$$F_F = v_i / f_i \times \left[1 / \left(1 - \text{Desirability of each state}\right)\right]. \tag{5}$$

Since velocity is directly, proportional to bat's closeness to a final state, and frequency of bat changes with respect to number of final states in the diagram, hence v_i remains in numerator and f_i is placed in the denominator. As for the desirability, a lower value indicates higher desirability. As its range lies between 0 and 1, it has been placed in the denominator. For optimisation purposes and to remove a few glitches found in the initial experiments, the desirability was subtracted from 1.

For example, if for one bat travelling from a state has $v_i = 98.51$, $f_i = 12.2$, Desirability of the state = 0.5, then $F_F = 10.07$.

Described next is the full-fledged algorithm for the proposed approach using bat algorithm.

The following section describes the step by step execution and working of the presented approach with an illustrated case study.

5 Experimental research

This section throws light upon how exactly the proposed algorithm works. This is shown by an example of an Automatic Teller Machine (ATM) system (http://www.math-cs.gordon.edu/courses/cps211/ATM Example/Statecharts.html). For each iteration of the algorithm, the changing parameter values and calculations are depicted. The algorithm works by minimising the objective function which gives complete state and transition coverage in optimal number of paths considering testing of critical portions of the state diagram at first. The algorithm has been tested for various other real world domain examples such as enrolment system, telephone model, class management system and many more. Following is the illustration of the ATM system.

Figure 2 describes the behaviour of the ATM system through a state diagram. In this, initially the customer is asked to enter the transaction details which is verified by the corresponding bank, if approved, then the transaction is completed and the receipt is printed, else if disapproved, the customer is asked to re-enter the correct details. The problem of invalid pin is also handled by the system itself.

The Proposed Algorithm
BEGIN
Step 1. Initialize the Bat population as bat_i (i=1,2,…,n)
Step 2. Initialize Velocity v_i, Loudness A_i, Location x_i, Frequency $\{f_{min}, f_{max}\}$
(Values suggested byYang, 2010)
f_{min}=1
f_{max}=10
$V_{i=}$1
A_i=100
Step 3. Repeat until (i<Maximum number of iterations)
totalNoOfTransitions is the total number of transitions in the state diagram.
transitionsTraversed is the total number of transitions currently covered by the bat.
While (totalNoOfTransitions > transitionsTraversed)
Step 3.1 Define Initial state as the current optimal state.
currentState = initialState
Step 3.2 Generate Bats according to out bounds of current optimal state [outbounds=fanout of the state]
Number of Bats = Number of outgoing transitions from current optimal state.
Step 3.3 Generate new solutions by adjusting frequency, updating Velocities and locations
R = randomWalk () (It is a function that assigns a random path to each bat for moving from current state to next state.)
Step 3.4 Select the local best solution among the generated solutions from Step 3.2 (Rank the bats) by applying the Fitness function [Eq. (5)]. This will now become the current optimal state.
F = fitnessFunction()
Fitness function gives the fitness of the solution. The solution with minimum value of the function is selected as the best one.
currentState = bat[nextState].presentState
currentState.loudness = bat[nextState].loudness
currentState.frequency = bat[nextState].frequency
currentState.velocity = bat[nextState].velocity
presentState is the target state for each bat
currentState is the optimal state chosen from the present state of all the bats.
nextState is the count of the state selected
Step 3.5 Goto Step 3.2 until
(current optimal state != final State)
Repeat until (!currentState.isFinal)
Step 3.6 Print the sequence of optimal states as a complete path. This will be the i^{th} ranked global best solution.
Step 3.7 end repeat.
All the test sequences are generated.
Step 4. Postprocess results and visualization
END

The states in the state diagram are abbreviated as follows: S1: Initial, S2: Getting Specifics, S3: Sending to Bank, S4: Completing transaction, S5: Printing Receipt, S6: Handling Invalid pin, S7: Asking customer if he wants another, S8: Final state 1, S9: Final state 2.

As discussed above, there are mainly two parameters used in the algorithm as inputs which are calculated as explained below,

- Desirability denotes the importance of the state which depends upon the number of its fan-ins and fan-outs.

  Desirability = fan-in × 0.75 + fan-out × 0.25, where this is in the range [0, 1] (‘0’ for maximum, ‘1’ for minimum)

- Probability of each state to reach the final state is taken as

  Probability = 1/Number of final states in the state diagram.

In Figure 2, S1 is the first current state. As S2 is the only reachable state from S1, this will be the optimal state and hence the next current state. Similarly, S3 is reached. From S3, the number of bats generated is 4[fan-out]. Each bat has following parameter values:

- α = 0.85(this is constant throughout used in equation (4) as the loudness decreases with a constant value (Yang, 2010)); f_{min} = 1, f_{max}= 10; Probability(to reach either final state) = 0.5
- *Bat 1*:

β =1.25

f_i = 12.25 from equation (1)

A_i = 61.41 from equation (4)

ε = 0.09 (random value used in equation (3))

x_{new}=15.50 (from Random Walk equation (3))

v_i = 158.51 from equation (2)

Fitness function F_F = 11.18 from equation (5).

Similarly, for other bats,

- *Bat 2*:

β = 1

f_i = 10.0

A_i = 61.41

ε = 0.05

x_{new} = 13.16

v_i = 121.80

Fitness function F_F = 11.44.

- *Bat 3*:

β = 1.75

f_i = 16.75

A_i = 61.41

ε = 0.72

x_{new} = 54.20

v_i = 833.20

Fitness function F_F = 47.24.

- *Bat 4*:

β = 0.75

f_i = 7.75

A_i = 61.41

ε = 0.61

x_{new} = 47.49

v_i = 379.91

Fitness function F_F = 48.52.

Among these possible next states, the value of fitness function is the minimum for state S4, hence it is the optimal state and next current state. Similarly, the values of fitness function can be calculated for other states and the complete path is generated.

Table 3 describes the sequence of one path starting from initial to final state. The path explained in this example covers self-loops (S6 → S6) and cycles (S7 → S2) to show the applicability of this algorithm. If a self-loop exists on a state, then this is given the highest priority otherwise either it will be left out or it will increase the repetitions. Hence, they need to be covered first (Iteration 8). When there are multiple transitions from a state (S3 and S7), the transition which is traversed will be blocked to avoid repetition of the same transition in next iteration (Iteration 3, 7 and 5, 10). Hence the path generated is S1 → S2 → S3 → S4 → S7 → S2 → S3 → S6 → S6 → S7 → S8.

Figure 2 UML state machine of an ATM transaction

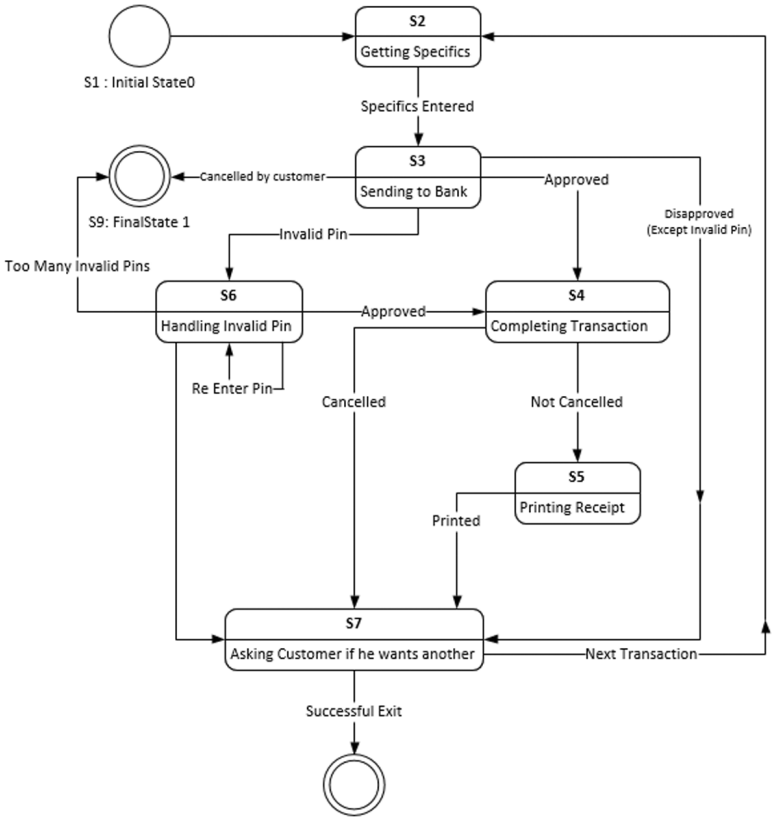


Table 3 Data table for one complete

Iteration	*Current state*	*Number of bats*	*Next states*	*Fitness function (F_F)*	*Optimal state*	*Location (x_{new})*	*Transition covered*
1	S1	1	S2	1.326	S2	1.96	S1 → S2
2	S2	1	S3	8.977	S3	9.61	S2 → S3
3	S3	4	S4	11.18	S4	13.16	S3 →S4
			S6	11.44			
			S7	47.24			
			S9	48.52			
4	S4	2	S5	35.64	S7	20.26	S4 → S7
			S7	11.87			
5	S7	2	S2	63.37	S2	57.26	S7 → S2
			S8	79.79			
6	S2	1	S3	295.92	S3	89.81	S2 → S3
7	S3	3	S6	76.73	S6	98.01	S3 → S6
			S7	79.33			
			S9	134.14			
8	S6	1	S6	87.06	S6	108.33	S6 →S6
9	S6	3	S4	125.56	S7	113.40	S6 → S7
			S7	67.34			
			S9	156.24			
10	S7	1	S8	286.38	S8	132.14	S7 → S8

Likewise, the algorithm generates all the possible sequences in this diagram and the same are mentioned here:

S1 → S2 → S3 → S4 → S5 → S7 → S8

S1 → S2 → S3 → S7 → S8

S1 → S2 → S3 → S9

S1 → S2 → S3 → S6 → S9

Therefore, the presented approach is capable of generating test sequences with full coverage, high optimality and minimum repetitions. Now, the efficiency of the algorithm can be compared and analysed with other prevalent algorithms which is covered in the next section.

6 Analysis

This section fulfils its purpose of analysing the empirical results of the presented approach along with a few previous techniques. The techniques suggested by Genetic Algorithm (Doungsa-ard et al., 2005), Ant Colony Optimisation (Srivastava et al., 2010a), Cuckoo Search Algorithm (Srivastava et al., 2012a) and the Intelligent Water Drop (Srivastava et al., 2012b) have already provided feasible solutions to many software testing problems.

The results of the proposed approach have been compiled and statistically tested with those of the above mentioned techniques, and they show promising results in continuing the research ahead.

Genetic algorithm (GA)-based approach of test sequence generation is applied by creating objective function for each path. GAs also individually generates subpopulations for each objective function. The transitions, which are traversed 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 (Doungsa-ard et al., 2005).

The ant colony optimisation approach (Srivastava and Baby, 2010) can also be used for test sequence generation using state machine diagram where 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. The proposed algorithm uses only certain parameters of the bat algorithm which alone can generate adequate sequences. Moreover, the number of equations involved in calculating the fitness function only increases the efficacy of the algorithm.

For each of the three algorithms, the experiments were carried out in conditions to deliver best results. For the Bat Algorithm, Yang has suggested to use values in the range as per the domain of problem (Yang 2010). Hence, the approach tries to stick to the ranges suggested and as the presented example does not belong to a large problem domain, the ranges of values are resized in proportion. Each result used in the calculation of coverage and nodal visits was an average of 25 experimental runs, only to reduce probability of stochastic results.

Even though the GA algorithm focuses on critical nodes of the diagram, it fails to address the complete coverage problem. On the other hand, the ACO approaches complete coverage in all diagrams it was run on, yet its repetition ratio is fairly poor. The IWD, such as ACO also holds its guarantee to deliver 100% coverage, but it loses in

deciding the critical sections which need be tested first. The Bat Algorithm-based approach flies higher than all these approaches by giving complete coverage, keeping in mind the critical sections of the diagram and also keeps a check on the repetition ratio.

Moreover, none of the approaches show any promise that they will work for state machines having multiple final states. The algorithm presented here runs and generates favourable results even when tested repeatedly on such state machines, having more than one end state.

Table 4 reflects the performance of the given approach with other existing approaches as mentioned in the table. It is clear that the proposed algorithm proves to be successful in providing complete code coverage as well as better repetition ratio than all other approaches.

The study in previous section shows interesting results and it is clearly visible that the proposed algorithm is efficient. The results of the proposed approach are compared with results of genetic algorithm and ant colony optimisation.

Also, statistical data about comparison between proposed algorithm, genetic algorithm, ant colony optimisation, cuckoo search and intelligent water drop is plotted in Figures 3 and 4.

Table 4 Table for comparative results of simulation

	Case study					
	Class management system		*Enrolment system*		*Telephone system*	
Algorithm	*Complete coverage*	*Repetition ratio*	*Complete coverage*	*Repetition ratio*	*Complete coverage*	*Repetition ratio*
Bat algorithm	Yes	1.5	Yes	2.14	Yes	2.07
Cuckoo search	Yes	1.67	Yes	3.0	Yes	2.11
Genetic algorithm	No	2.33	No	4.43	No	2.89
Ant colony	Yes	3.0	Yes	5.43	Yes	3.1
Intelligent water drop	Yes	2.6	Yes	2.9	Yes	2.13

Figure 3 Comparison of algorithms on number of nodes visited in one complete run

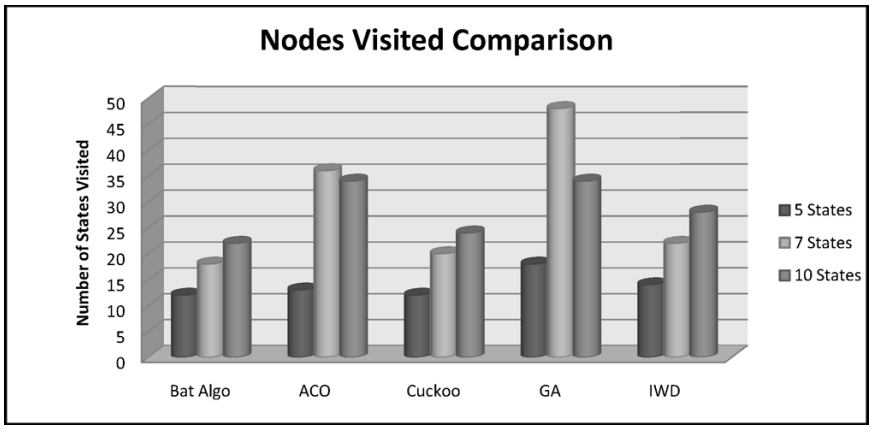


Figure 4 Growth of algorithm w.r.t nodes

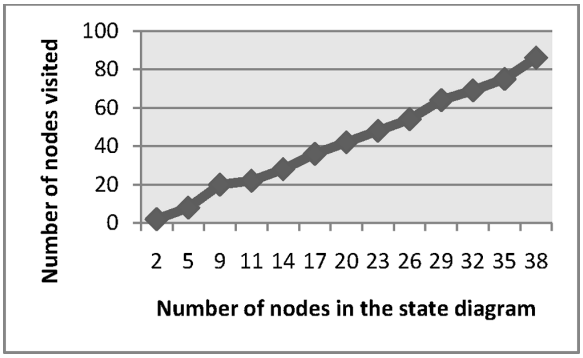


Figure 3 shows the comparison of the number of the nodes visited by the many approaches when applied to same state diagrams. The optimal transitions are obtained when all transitions can be covered with least number of states visited. The bat algorithm-based approach shows much better results than all of the techniques. Only Cuckoo search-based approach gives results comparable with the presented approach.

Figure 4 shows the graph between the numbers 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 the algorithm does not grow exponentially with increase in number of states.

7 Drawbacks and future work

Randomisation has been included for deriving the next state from the current state for each bat through random

walk, this produces different paths after obtaining the longest path at every execution. But, it also ensures total path coverage at the same time. As a future work, generation of test sequences will be worked upon with consideration of more guard conditions for faster convergence. Also, work can be extended for state machines having huge number of states and analysis using different statistical techniques.

8 Conclusion

The presented approach implemented a variation of the bat algorithm used to tackle meta-heuristic problems. This approach was tested on parsed XML files of respective UML state machine diagrams for which the algorithm attempts to generate optimal test case sequences and achieve complete coverage.

As per the statistics given and tests conducted, the proposed approach yields results better than existing techniques of genetic algorithm, ant colony optimisation techniques, intelligent water drop and cuckoo search algorithm in terms of coverage and optimality of generated sequences. This approach can readily be implemented and easily replace present techniques for sequencing industrial test cases.

References

Alander, J.T., Mantere, T. and Turunen, P. (1998) 'Genetic algorithm based software testing', *Proceedings of the 3rd International Conference on Artificial Neural Networks & Genetic Algorithms*, Wein, Austria, pp.325–328.

Altringham, J.D. (1996) *Bats: Biology and Behaviour*, Oxford University Press, USA.

Blaha, M.R. and Rumbaugh, J.R. (2011) *Object Oriented Modeling and Design with UML*, 2nd ed., Pearson, USA.

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

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

Gandomi, A.H., Yang, X.S. and Alavi, A.H. (2011) 'Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems', *Engineering with Computers*, p.27.

Janert, P.K. (2002) *Simple XML Parsing with SAX and DOM*, O'Reilly Publications, USA.

Li, H. and Lam, C.P. (2004) '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.

Luke, S. (2012) *Essentials of Meta-Heuristics: A Set of Undergraduate Lecture Notes*, Rev C, Online Version 1.3, Department of Computer Science, George Mason University.

Mayers, J.G. (2004) *The Art of Software Testing*, 2nd ed., John Wiley & Sons, Hoboken, NJ.

Pressman, R. (2001) *Software Engineering–A Practitioner's Approach*, 5th ed., McGraw-Hill, New York, NY.

Rathore, A., Bohara, A., Gupta, P.R., Lakshmiprashanth, T.S. and Srivastava, P.R. (2011) 'Application of genetic algorithm and tabu search in software testing', *Proceedings of the Fourth Annual ACM Bangalore Conference*, pp.2–4.

Srivastava, P.R. (2010) 'Structured testing using an ant colony optimization', *IITM 2010*, ACM ICPS, Allahabad, December, pp.205–209.

Srivastava, P.R. and Baby, K. (2010) 'Automated software testing using metaheuristic technique based on an ant colony optimization', *Proceedings of International Symposium on Electronic System Design (ISED)*, IEEE Computer Society, pp.235–240.

Srivastava, P.R., Agarwal, K. and Goyal, M. (2012b) 'Code coverage using intelligent water drop', *Int. J. Bio-Inspired Computation*, UK.

Srivastava, P.R., Jain, P., Baheti, S. and Samdani, P. (2010b) 'Test case generation using genetic algorithm', *International Journal Information Analysis and Processing*, Vol. 3, No. 1, pp.7–13.

Srivastava, P.R., Jose, N., Barade, S. and Ghosh, D. (2010a) 'Optimized test sequence generation from usage models using ant colony optimization', *International Journal of Software Engineering and Applications*, Vol. 1, No. 2, pp.14–28.

Srivastava, P.R., Singh, A.K., Kumar, H. and Jain, M. (2012a) 'Optimal test sequence generation in state based testing using cuckoo search', *International Journal of Applied Evolutionary Computation*, Vol. 3, No. 3, pp.17–32.

Yang, X.S. (2010) *A New Metaheuristic Bat-Inspired Algorithm*, Nature Inspired Cooperative Strategies for Optimization, Spain.

Yang, X.S. and Deb, S. (2009) 'Cuckoo search via Lévy flights', *Proceedings of the World Congress on Nature & Biologically Inspired Computing*, pp.210–214.

View publication stats