

Apply Improve Monkey Algorithm (IMA) for Software Testing Process

Authors: Alaa Ibrahim Mahmood¹; Ibrahim Ahmed Saleh²

Directorate of Education in Nineveh, Mosul, Iraq¹

Department of Software, College of Computer and Mathematics, University of Mosul, Iraq²

alaaibmah2@gmail.com¹, i.hadedi@uomosul.edu.iq²

ABSTRACT

Software testing process is very important phase during development of software life cycle. In software testing, cost and improve efficiency is popular parameters for testing process. This paper is applied testing process by using novel optimization method based on combined between greedy and monkey algorithms (MA). The fitness function based on Markov decision model to simplify by a reduction testing states. Basic monkey algorithm was improved by using excellent solution of the greedy algorithm.

The optimal solution was obtained from MA. Simulation results show that solution to enhanced test method. The proposed algorithm is improved testing method give shortest time and optimum cost of testing software. Structures of paper are as follows. Section 1 including introduction, Section 2 literature reviews for the previous works, Section 3 include software testing model, section 4 represents proposed greedy algorithm solution. Section 5 proposed monkey algorithm for case selection, section 6 describes applied and analysis use case testing for algorithm, finally section 7 concludes the paper.

Keywords: greedy algorithm, markov's decision, monkey algorithm, software testing.

1. INTRODUCTION

Software testing is process of examining, analyzing software tools, and related documentation to identify to improve product quality. The main goal of testing is evaluated quality of an application, to improve software, reliable, detect software failures and defects may be discovered and corrected [1].

Modern software is being applied indifferent fields such as space, nuclear power....etc. This has become an urgent need for software quality issues and more important energy comprehensive attention. As one of

important means to ensure software quality, testing process plays an important role in the program life cycle [2]. In software testing, there is a principle that universal testing is impossible based on effective test of system under test. Best test experimental groups reduce cost of testing programs [3]. The main idea is improve software by looking for an execution sequence of a better test case which effect to software [4]. Practice results show that using a good test for execution sequences, it can greatly reduce cost of software testing [5-7].

In research [8], deal with selection strategy of test cases from perspective of control, treated software test as a control problem, regarded software under test as control object and selected test case as the corresponding controller. It tested abstracted into a Markov decision process, and used dynamic programming method to optimize software test. The dynamic programming method can get exact solution, but as software scale increases, the heuristic algorithm is needed to get optimal solution or asymptotic optimal solution. In smaller software, dynamic programming methods can get accurate solutions, but the software scale increases, heuristic algorithms are needed to get the optimal solution or the most asymptotic problem. One of solution used monkey algorithm (MA) optimization algorithm which based on behavior of monkeys when searching for source of food and formatting [9].

Monkey algorithm (MA) is used for software test case depend on the simulation of mountain-climbing processes of monkeys. It consists of three processes: climb process, watch-jump process, and somersault process [10]. It can effectively reduce the test cost, but this method also has large scale search are easy to fall into local optimum[11]. For a larger scale software system, its demand will be very large and

corresponding to demand many of test cases is even greater. In this case, use of the basic monkey algorithm reduced search efficiency. For this reason used greedy algorithm to choose the best test case currently to reduce the cost of testing a large unit [12].

In this paper the idea of Monkey and greedy algorithm[13,14] was used in software testing called "improved monkey algorithm IMA". To test software, first select necessary test cases from set of test case, and then remove the requirements covered by total demand, and this will not cover the remaining redundancy tests for aggregate demand to remove with the receipts, reducing size of test case set. To reduce test case set a greedy algorithm can be used to obtain a solution, and added the result to monkey's search algorithm of all nodes to the solution, so that monkey algorithm can find obtained optimal solution.

2. LITRATURE REVIEW

There is several research approaches that have been proposed which attempted to Software testing there are some of them:

In 2011, Wang Jun and other researchers suggested style for test case priority using genetic algorithm, the researcher was depended to regression-based testing was enhanced in the manner that faults can be detected more quickly, that lead to increase the Software Reliability [15]. In same year AdiSrikanth and others proposed honey bee optimization to reduce the software test in case study layer in software approach. Although the given good result but it have long time to test [2]. In 2012 Tajinder Singh and Mandeep Kaur Sandhu proposed many techniques are used in the software testing environment using hybrid technical algorithm that combine bee colony optimization with genetic algorithm, the paper describes the framework of software under test (SUT). The result of testing which is efficient take less time as compare to the existing GA[16]. In 2015, L.S. de Souza, et al. [17] presented two algorithms, First one was Multi-Objective particle swarm optimization MOPSO and other was Crowd in Distance-Roulette Wheel in field of Genetic algorithm, the two algorithms combined with the Harmony Search that was utilized in GA the selection of test cases. The goal of test case selection was to search the pertinent test case subset based on the condition of implementing test tolerability. The method showed better performance that MOPOS is best for test case

because the execution cost of the method was low. An investigation of hybrid strategies was performed on a smaller number of programs only.

In 2016 Kumar, et al. proposed also bee colony optimization to test data generation in some performance evaluation such as Condition type and weights the test is apply to some standard program, the researchers were reached to that there algorithm is best from other methods[18].

In 2017, M. Khari, et al. [19] introduced enhancement of tools with significant components for software lifecycle. The two components of software testing were test suite optimization as well as generation. The proposed method was able to supply a set of least test cases with maximum path coverage. The automated fault detection was performed by generating optimal test suite. The automated testing model included bee colony optimization (BCO) and Common Scrambling algorithm both separately, and then the hybrid two algorithms for given better results in every aspect and the algorithm can use large amounts of input data.

In 2019, L. Pavithra and S. Sandhya were introduced paper deal with Survey on Software Testing, the paper also test different levels of software testing. They describe a comprehensive account of all parameters and nomenclature that are belong to software testing[20].

3. PROPOSED APPROACH

The proposed software testing system is designed and implemented based on monkey's search and greedy algorithms, the system are finalize application software in user requirements which explained in detail in following steps.

3.1 Software Markov Decision Model

In order to describe proposed system for software testing, it used (KDD) dataset cup contain 41 procedure include basic features of individual TCP connections [21]. The testing process applied a Markov's decision model which define tuple $M(S,A,N,Q)$. After each test case is executed, it will cover some software requirements. The number of software requirements has been covered as the state of decision time t . Each test moment is selected to execute a test case. The state of each decision time depends on the previous decision. The state of the moment and the selected test case, so that the entire software testing process constitutes a Markov decision,

as shown in figure(1).

S:finite state A: action N :finite set of action
Q:Reward function

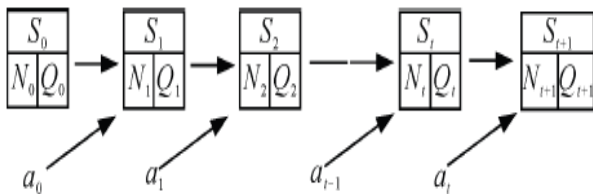


Fig 1: Markov decision process

For some characteristics of software testing, make test software such as some assumptions:

1. Use R_t to represent number of software requirements that have been covered at time t quantity, the total demand of system under test is n , the initial time $R_0 = 0$;
2. The number of times for software requires covered at each decision time t ($t = 0, 1, 2, \dots$) is taken as state S_t of system at time t , ie $S_t = R_t$
3. Only one test case should be selected at each decision time. The number of requirements that test case a_i can cover at time t is R_{ai/S_t} ;
4. After test case is executed, the corresponding requirement of test case is considered to be over written. If the system state at time $t - 1$; $S_{t-1} = R_{t-1}$, then system state $S_t = R_t = R_{t-1} + R_{ai/S_t}$ at time t .
5. The target state at end of test is $S_t = R_t = n$;
6. The cost of executing test case a_i at time t is $W_t = \text{cost}(a_i)$;
7. If τ is used to indicate time, it take for system state to reach target state, see equation 1
Where: ω represents order in which test cases are executed; E_ω means cost required for ω is expected execution order; J_ω means tested expected value of

$$J_\omega = E_\omega \sum_{t=0}^{\tau} W_t \dots \dots \dots (1)$$

$$x_{ij} = \begin{cases} 0 & \text{First test case has no relationship with first requirement} \\ 1 & \text{First test case has a relationship with the first requirement} \end{cases}$$

total cost is generated in the process.

3.2 Software Test Parameter

Construction

The initial parameters of software testing are $A = \{a_1, a_2, a_3, \dots, a_m\}$, to represent set of test case, then $R = \{r_1, r_2, r_3, \dots, r_n\}$ represent set of collection requirements [12]. The relationship between test requirement set and test case set is represented by X . The matrix has m rows and n columns representing m test cases and n requirements, the element values of the matrix are:

Each row of relation matrix corresponds to a non-negative cost $\text{cost}(a_i)$, which represents the labor time and other cost of each test case execution. The goal of testis selected one test case sequence that will cover requirement set R with lowest cost. Following definitions represent test cases:

Definition 1:(Essential test case),for test case a_p If and only if test case a_j coverage requirement r_i in alternative use case set A , then a_j is an essential test case.

Definition 2 :(Redundant test case) ,for redundant test case a_j , if and only if the demand set R does not contain the requirement covered by test case a_j [21].

According to the relationship matrix X of the test requirement set and test case set, select the essential test case set $A_{bi} = \{a_1, a_2, \dots, a_p\}$ from test case set A , where $0 \leq p \leq m$, and from The test case set is removed. The k requirements covered by test set A in the requirement set R are removed from R to obtain a new demand set R' . Then, according to new requirement set R' , the redundant test case set $A_{rong} = \{t_1, t_2, \dots, t_q\}$ is found from test case set A , where $0 \leq q \leq m$, and $0 \leq p + q \leq m$, and remove it from test case set. Finally, a reduced scale test case set A' is obtained.

Definition3:(Unit cost), requirement coverage for test

$$C_{R_j/S_t} = \frac{R_{a_i/S_t}}{\text{cost}(a_i)} (2)$$

cases $C_{Raj/St}$. Demand coverage is number of requirements covered by test cases and ratio of total demand can be expressed as in Equation 2:

During testing process, as testing continues, as requirements are constantly changing, the number of requirements covered by each test case is also changing, resulting in a unit cost requirement coverage rate $C_{Raj/St}$ also with test and constantly changing. Therefore, after each test case is executed, according to test results, the unit cost requirement coverage rate $C_{Raj/St}$ of each test case is calculated in real time online.

4. GREEDY ALGORITHM SOLUTION

The basic idea to use a greedy algorithm in proposed work is to choose best current test case for each choice, where best test case currently uses test case with a large unit of cost coverage [11], see Equation 3:

$$a_i = \operatorname{argmax} \frac{R_{a_i/S_i}}{\operatorname{cost}(a_i)} \quad (3)$$

The steps of search process in framework of greedy search are[11]:

1. Select test cases with large unit cost coverage from test case set,
2. Update the relevant parameters during software testing process,
3. Repeat 1 and 2 until demand's set is fully covered.

The specific algorithm is as follows:

- **Compute($C_{Raj/St}$)**// Calculate unit cost requirement coverage of each test case under state S_t //
- **Input:** $t-1$ time coverage requirement for all test cases ($R_{aj/St-1}$) and unit cost requirement coverage rate ($C_{Raj/St-1}$).
- **Output:** t time unit cost requirements for all test cases Cover rate $C_{Raj/St}$.

Begin

Update(R_{St})//**Update total number of requests, upcoming test//**

The requirements covered by trial case A_{t-1} are

removed from the total demand

$$\left\{ \begin{array}{l} R_{S_t} = R_{S_{t-1}} - R_{A_{t-1}/S_{t-1}} \end{array} \right\}$$

For $j = 1$ to l //**The test case A_{t-1} that will be executed//**

The covered requirements are moved from remaining test cases that have not been executed.

Divide and calculate the unit cost requirement coverage:

$$\text{if } R_{a_t/S_{t-1}} \cap \frac{R_{A_{t-1}}}{S_{t-1}} \neq \emptyset$$

5. MONKEY ALGORITHM (MA)

FOR CASE SELECTION

The basic idea of applying monkey algorithm in case selection is covered optimal transition to solve problem through construct climbing on tree or mountain to search for food either individually or in groups. MA consists of three processes: climb process, watch-jump process, and somersault process.

- **Climb process** used to search for local optimal solutions.
- **Watch-jump process** used to look for other points, whose objective values exceed current solutions so as to accelerate the monkeys' search courses.
- **Somersault process** to make monkeys turn up to new search domains rapidly.

After applied MA[22], uses values on directed path to represent possible solution, and converts objective function value of problem to values on construction path. The problem is transformed into solving optimal values problem on construction diagram [23]. Usually monkey algorithm includes in the following steps:

1. The feasible created by look for on solution space parameterized probability distribution model;
2. Updating the parameterized probability model by using solution generated by search, updating probability distribution parameter of solution space, so that search on new model can be focus in high-quality solution.
3. If the algorithm reaches stop condition, stop and

save optimal solution; if not, go to (1).

Although monkey algorithm has many advantages, the Initialization of population will have great effect on precision, which leads to monkey's randomness in process of starting search process. The climb process is a step-by-step procedure to change positions of monkey from initial positions which lead the algorithm search is low and search takes a long time. For these reasons, greedy algorithm is used to solve the problem and enhance test case with large coverage and low cost.

- The initial parameters increase probability of being selected is (to $N=5$, alteration=500, position of each monkey = x_i for $i=1,2 \dots N$, Perturbation rate=0.1)
- Establishing the search on a better basis, thereby reducing time spent searching.
- Set the number of monkey
For each monkeys groups: *monkey* = 1, 2... N, number of test cases.
- The scaled test case set A' is $m - p - q$, which is

$$\left\{ \begin{aligned} R_{a_t/S_{t-1}} &= R_{a_j/S_{t-1}} - \left(\frac{R_{a_t}}{S_{t-1}} \cap \frac{R_{A_{t-1}}}{S_{t-1}} \right); \\ C_{R_{a_j/S_t}} &= \frac{R_{a_j}}{S_{t-1} \cdot \text{cost}(a_i)} \end{aligned} \right\}$$

implemented in greedy algorithm in section 3.

- The number of test cases is o , and initial values $\tau_0(S_b, a_i) = h$ corresponding to test case node is selected in the greedy algorithm.
- The initial value corresponding untested test case node in the greedy algorithm is still $\tau_0(S_t, a_i) = 1$.
- Where h is coefficient is enhanced by selected test case node.
- If h is too large, it is easy for algorithm to fall into local optimization of greedy algorithm;

- If h is too small, the effect of initial information enhancement is not obvious. After many attempts, chose $h=1.2$.

$$\Delta\tau_k(S_t, a_i) = \sum_{j=1}^j \Delta\tau_k^j(S_t, a_i) \tag{5}$$

To improve monkey algorithm is by enhancement currently found optimal values. In the k_{th} iteration, after all monkeys complete a traversal, the test costs of all monkeys are sorted, and p_r monkeys are released before test cost is small. The monkeys $M_j(i=1,2,\dots,N)$ released by different states S_i and its corresponding selection test case a_i is, see Equation 4 .

$$\Delta\tau_k^i(S_t, M_j) = \begin{cases} \frac{Q}{V_{\pi(i)}} \text{ where } \pi_k^i = M_j \\ \dots \\ 0 \text{ otherwise} \end{cases} \tag{4}$$

Where: Q is a constant given in advance. The rest of monkeys do not release and each iteration produces, see equation 5.

The monkey values update rules after each iteration, in climb process states [24]:

After the climb process, each monkey arrives at its own mountaintop. And then it will take a look and determine whether there are other points around it being higher than current one. If yes, it will jump there from current position. The positive parameter b as eyesight of the monkey which indicates to maximal distance that the monkey can watch better. The probability π may be calculated, see equation 6:

$$P_k(S_t, a_i) = 0.9 * \frac{\tau_k(S_t, a_i)}{\max(\tau_k(S_t, a_i))} \tag{6}$$

to obtain M solutions, where M is initial population for number of monkeys or agents. Each ones of these monkeys has a position is given by vector x . In order to place a monkey is necessary to define possible initial positions, which can be an analogy of a jungle; this jungle represents software test cost. Monkey positions are generated at random verifying that monkey position is feasible; if not, a new set of positions will be generated until a feasible position is produced. The pseudo-code of MO algorithm is shown in figure (2):

```

Step1: Initialization
population of M monkeys is generated randomly in the feasible space
with the position  $X = (x_1, x_2, \dots, x_D)$ ,  $i = 1, 2, \dots, M$ ,
where D is the dimension of the problem.
Define low and high of Xi
Assign values to parameters: a, b, c and d

Step2: Climb process loop
For I = 1: I_max (max. iteration)
  For counter l= 1: Nc (number of cycles)
    Randomly generate vector  $\Delta X(I)$ 
    Generate the pseudo-gradient of the objective function  $f'(X_i)$ 
    Generate new monkeys' position  $Y_i$  using a and  $f'$ 
    Update  $X_i$  with  $Y_i$  if feasible
  End For Nc
Step3: Watch-jump process
Generate new  $Y_i(I)$  from  $(X_i-b, X_i+b)$ 
If  $-f'(Y_i) \geq -f'(X_i)$ 
  Update  $X_i(I)$  with  $Y_i(I)$  if feasible (i.e., within limits)
End If
Apply climb process loop again
Rank the positions and find current best solution so far
Estimate Pivot P(I) from  $X_i$ , c and d
Calculate  $Y_i(I)$  around the pivot
Update  $X_i(I)$  with  $Y_i(I)$  if feasible and repeat somersault until feasible
Rank the positions and find current best solution
If (the new monkey's position violates its boundary limits)
  Use Search-based function to handle constraints
End If
End For I
Return the highest mountaintop ( $-f(X)$ ) and its position X
    
```

Fig 2: Pseudo code for MA

6. EXPERIMENTS RESULTS

To test the efficiency of proposed work for software test case we used a 18 test requirements named **R**, where $R = \{r_1, r_2, \dots, r_{18}\}$, and select 26 test cases from use case library to form a test case set $A = \{a_1, a_2, \dots, a_{26}\}$. The **R** can represent either all of program's test case requirements or those requirements related to program modifications[25]. A representative set of test cases that satisfy, the **R** must contain at least one test case from each T_i . The requirements covered by each test case are as follows, see Table 1:

Table 1. Test cases cover demand situation

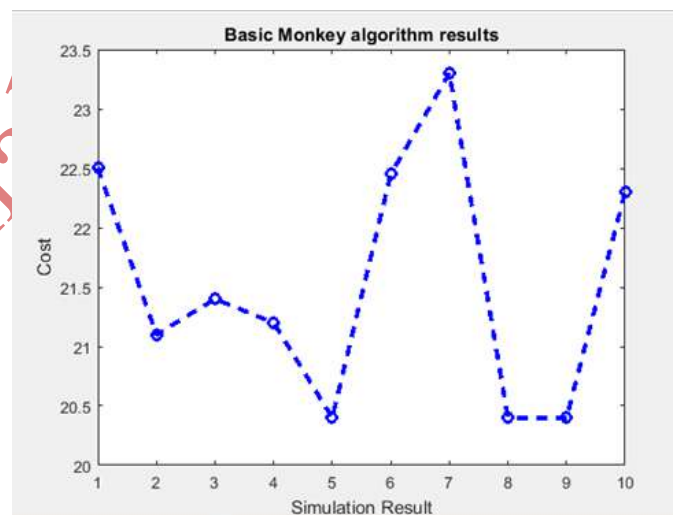
No.	Test case	Covered demand	Cost	No.	Test case	Covered demand	Cost
1	t ₁	r ₃	2.38	14	t ₁₄	r ₇	2.11
2	t ₂	r ₈	1.73	15	t ₁₅	r _{10, r₁₃}	2.05
3	t ₃	r ₁₈	2.75	16	t ₁₆	r _{4, r₆}	2.91
4	t ₄	r ₂	2.50	17	t ₁₇	r ₉	2.03
5	t ₅	r ₄	2.04	18	t ₁₈	r ₁₀	2.73
6	t ₆	r _{6, r₁₄}	1.76	19	t ₁₉	r ₄	2.12
7	t ₇	r _{7, r₁₇}	1.78	20	t ₂₀	r _{1, r₁₁}	1.58
8	t ₈	r ₈	1.06	21	t ₂₁	r ₁₃	2.98
9	t ₉	r ₂	2.7	22	t ₂₂	r _{2, r₆}	1.05
10	t ₁₀	r _{2, r₈}	2.61	23	t ₂₃	r _{5, r₁₅}	2.92
11	t ₁₁	r ₄	2.39	24	t ₂₄	r _{14, r₁₈}	2.48
12	t ₁₂	r ₉	1.4	25	t ₂₅	r ₁₇	1.03
13	t ₁₃	r _{9, r₁₂}	1.86	26	t ₂₆	r _{2, r₁₂}	1.87

From Table 1, it can see basic monkey algorithm is used to solve problem. 10 simulations operations are performed. The simulation results are shown in figure (3). It is seen from simulation results that optimal solution obtained by basic monkey algorithm is different for each simulation, indicating larger sample size makes MO algorithm easily fall into local optimal solution during optimization process.

Fig 3: Basic Monkey algorithm results

Then use test case reduction method discussed to find essential test cases $A_{bi} = \{a_1, a_{13}, a_{20}, a_{23}, a_{26}\}$, and after updating essential test cases, update total requirements and test case coverage. Requirements, find redundant test cases $A_{rong} = \{a_4, a_9, a_{12}, a_{17}\}$, and reduce test case set. The reduced test case set scale is reduced from 26×18 to 17×11

After a greedy algorithm is performed on reduced test case set and obtain a better test case sequence in test set {a8, a6, a15, a7, a5, a24} from necessary total cost .The price is J = 21.78. Although the solution



obtained by greedy algorithm is same every time, compared with solution obtained by the basic monkey algorithm, it can be found solution obtained by greedy algorithm is not optimal.

To perform improved monkey algorithm (IMO). The initial parameters of six test cases selected by greedy algorithm are added to monkey algorithm by theoretical description of test method above. On this basis, optimal test set obtained by monkey algorithm is {a16, a7, a15, a8, a24}, plus necessary test. The simulation results show the improved monkey algorithm can gave good local optimum, and shorten search time as shown in figure (4).

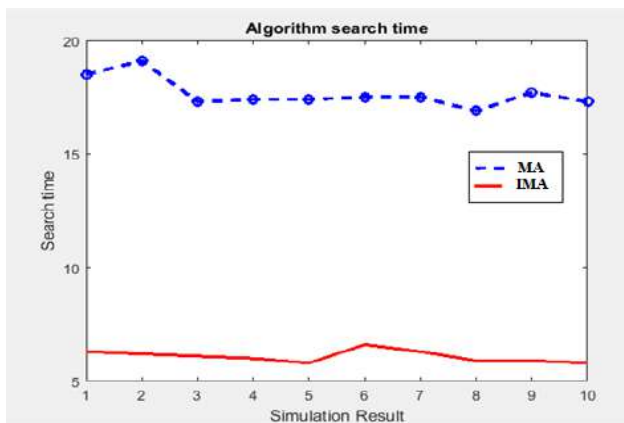


Fig 4. Algorithm search time

As in figure (4), that seen search time for requirement observe rate of search time in improved monkey algorithm (IMO) is about 1/3 of basic monkey algorithm (MA). The improved monkey algorithm indicates that not only has a good search effect, but also can greatly reduce search time and improve the algorithm effectiveness.

7. CONCLUSIONS

In this paper proposed techniques applied optimize test case sequence through comparison of basic monkey algorithm (MO) and heuristic algorithm combined monkey and greedy algorithm called "Improved Monkey algorithm (IMO)", it used KDD data has set with 41 attributes is used in this empirical study. Although for monkey algorithm has optimal solution but the initial time is random and spends long time that leads to combine greedy algorithm to select best current test case for each choice that lead to reduce initial time. Improve monkey search verifies effectiveness of method through simulation analysis. The simulation results show proposed method is more efficient than basic monkey algorithm. IMO can reduce test cost of regression testing and improve efficiency of software with optimized test suite. In addition to, in the future we plain to work depends on automation testing used to re-run test scenarios used emperor penguin algorithm that were performed quickly and repeatedly.

8. REFERENCES

[1] k. D. R. Dhivya, V. S. Meenakshi and B.

Priyadarshini, "Analysis on Generating Test Cases for Random Testing Using Optimization Techniques," *International Conference on Information Technology & Society* 8-9 June 2015, Kuala Lumpur, Malaysia, 2015, pp.200-207.

[2] Adi Srikanth, N.J. Kulkarni and K. V. Naveen, "Test Case Optimization Using Artificial Bee Colony Algorithm," *Conference Paper in Communications in Computer and Information Science*, July 2011, pp. 570-579.

[3] J. Zuo, Z. Zhonghua, W. Qiming, C. Yazuo, "A Requirement-based Approach to System-level Black-box Test Case Prioritization," *Computer and Digital Engineering*, vol. 38, issue. 5, pp. 68-73, 2010.

[4] S.M.K. Quadri, S. U. Farooq, "Software Testing-Goals, Principles and Limitations," *International Journal of Computer Applications*, vol.6, issue.9, pp.7-10, 2010.

[5] T. Hong, Y. Tian-yun, L. Jun-Qiu, "New Algorithm for Attribute Reduction Based on Intrusion Detection," *Science Technology and Engineering*, vol. 8, issue. 18, pp. 5289-5292, 2008.

[6] Y. Guang-hua, B. Yang, L. Dong-hung, T. Le-le, "Test case prioritization based on requirement," *Computer Engineering and Design*, vol.32, issue.8, pp. 2724-2728, 2011.

[7] B. Suri, and S. Singhal, "Implementing Ant Colony Optimization for Test Case Selection and Prioritization," *International Journal on Computer Science and Engineering (IJCSE)*, vol.3, issue 5, pp.1924- 1932, 2012.

- [8] P. Hong, "Research on intrusion detection method based on rough set," *Electronic Science Journal of Technical University*, vol.35, issue.1, pp. 108-110, 2006.
- [9] X. Chen, Y. Zhou, Q. Luo, "A Hybrid Monkey Search Algorithm for Clustering Analysis," *Hindawi Publishing Corporation the Scientific World Journal*, vol.2014, Article ID 938239, 16 pages <http://dx.doi.org/10.1155/2014/938239>.
- [10] R. V. Devi and S.S. Sathya, "Monkey Behavior Based Algorithms - A Survey," *I.J. Intelligent Systems and Applications*, vol. 12, pp. 67-86, 2017.
- [11] Z. Dong, N. Liu, R. Rojas-Cessa, " Greedy scheduling of tasks with time constraints for energy-efficient cloud-computing data centers," *Dong et al. Journal of Cloud Computing: Advances, Systems and Applications (2015) 4:5*, pp. 1-14, DOI 10.1186/s13677-015-0031.
- [12] S. Barman, S.K. Krishnamurthy, R. Vaish, "Greedy Algorithms for Maximizing Nash Social Welfare," pp. 1-13, 2018.
- [13] W. Jun, Z. Yan and J. Chen, " Test Case Prioritization Technique based on Genetic Algorithm," *2011 International Conference on Internet Computing and Information Services*, Hong Kong, 2011, pp.173 – 175.
- [14] S. M. McGovern and S. M. Gupta, "Greedy Algorithm for Disassembly Line Scheduling," *Proceedings of the 2003 IEEE International Conference on Systems, Man, and Cybernetics*, Washington, DC, pp.1737-1744, October 2003.
- [15] S. Sampath, S. Sprenkle, E. Gibson, L. Pollock and A.S. Greenwald, "Applying Concept Analysis to User-Session-Based Testing of Web Applications," in *IEEE Transactions on Software Engineering*, vol. 33, no. 10, pp. 643-658, Oct. 2007, doi: 10.1109/TSE.2007.70723.
- [16] T. Singh, M.K. Sandhu, " An Approach in the Software Testing Environment using Artificial Bee Colony (ABC) Optimization," *International Journal of Computer Applications (0975 – 8887)*, vol. 58, issue. 21, November 2012.
- [17] L. S. de Souza, R.B.C. Prudenico, F.A. de Barros, "A hybrid particle swarm optimization and harmony search algorithm approach for multi-objective test case selection," *Journal of the Brazilian Computer Society*, vol.21, issue.19, pp. 1-20, 2015.
- [18] S. Kumar, D. K. Yadav, D. A. Khan, " Artificial Bee Colony based Test Data Generation for Data-Flow Testing," *Indian Journal of Science and Technology*, vol. 9, issue.39, pp. 1-10, 2016, DOI : 10.17485/ijst/2016/v9i39/100733.
- [19] M. Khari, P. Kumar, D. Burgos, R.G. Crespo, "Optimized test suites for automated testing using different optimization techniques," *Soft Computing*, vol. 22, issue. 24, pp. 8341-8352, 2018.
- [20] L. Pavithra, S. Sandhya, "Survey on Software Testing," *Journal of Network Communications and Emerging Technologies (JNCET)*, vol. 9, issue. 3, 2019.
- [21] P. Aggarwal and S.K. Sharma, " Analysis of

KDD Dataset Attributes - Class wise For Intrusion Detection," *3rd International Conference on Recent Trends in Computing 2015 (ICRTC-2015)*, *Procedia Computer Science* 57, 2015, pp. 842 – 851.

[22] C. Lin, J. Liu, C. Ho, "Anomaly Detection Using LibSVM Training Tools," *International Journal of Security and Its Applications*, vol.2, issue.4, pp. 89-98, 2008.

[23] IA Saleh, OI Alsaif, KH Thanoon," Deep Coverage Strategy for Private Wireless Network Power Using Hybrid (Salp Optimization–Genetic) Algorithms," *Technology Reports of Kansai University* ,vol. 62, issue. 3, pp. 45-54, 2020.

[24] KI Alsaif, IA Saleh, OI Alsaif "Design and Analysis WLAN Nodes Performance Based on Roaming Technique".*Journal of Education for Pure Science* Vol. 4, issue. 1, pp. 228–235, 2014.

[25] Y. Qingqing, S. Huairong, S. Qionglng, "Ground-based Space Surveillance Rescheduling Based on Ant Colony Optimization," *Ordnance Industry Automation*, vol.35, issue.3, pp. 1-5, 2016.