Swarm Intelligence Techniques and Genetic Algorithms for Test Case Prioritization

Tina Sachdeva

Abstract: ***Regression testing is a technique which is carried out to ascertain that the changes that were done in the source code have not negatively damped its performance. Hence, it is a crucial and an expensive step of the software development life cycle. It re-establishes confidence in correctness of the software after changes were made to it. A test suite is used to test the software, but often it becomes time consuming to re-execute each test case every time regression testing is done. Therefore, it becomes essential to decrease the number of the test cases by prioritizing them based on some criterion. This ensures maximum detection of faults in least amount of time. In this paper, author has compared swarm intelligence techniques with genetic algorithms for such a test suite prioritization. In particular, by taking a sample GCD program Ant Colony Optimization (ACO) has been compared with Genetic Algorithms (GA) for the purpose of test suite minimization. Unit of comparison has been execution time required for prioritization of test cases. Further, experimental results have been compared with time taken by both with random testing.***

Keywords: Test Case Prioritization, Regression Testing, Swarm Optimization, Ant Colony Optimization, Bee Colony Optimization, Genetic Algorithm

I. INTRODUCTION

Software Testing Life Cycle defines the steps to be employed in testing the software and regression testing is a crucial phase in this. Changes may be made to the application code if a new feature is added or the user demands a change in the requirements. Such enhancements may cause the software to function in an undesirable manner. In order to fix the software and ensure its smooth functioning, regression testing is used. Its purpose is to ensure that the changes have not introduced new faults in the software. Regression testing uses one of the following three approaches:

i. **Retest All**: Each test case needs to be executed.
ii. **Test Selection**: This technique runs a part of the test suite to satisfy cost constraints.
iii. **Test Case Prioritization**: Order test cases in the suite in order to maximize the number of faults detected in least time.

However, running all test cases is not feasible every time, considering the resource limitations. In such a scenario, test case prioritization is the most common of the three approaches stated.

Test suites play a prominent role in software testing. Test case is a data set, such as an input data, execution paths, execution conditions, or testing requirements. Problems associated with test suite are high volume of test case number, high manpower cost, test case coverage, etc. Therefore, efficient management of test suite is a potential research problem in the domain of software testing.

During recent times, novel approaches of Swarm Intelligence have begun to be used for prioritization of test cases. Genetic Algorithms have also found their way into it. We studied the recent developments in these directions i.e., the use of Swarm Intelligence Techniques, particularly, Ant Colony Optimization and compared its performance with Genetic Algorithms. Also, some improvements suggested to the traditional Ant Colony Optimization to enhance its performance and efficiency have also been discussed in this paper.

II. RELATED WORK

Over the years, many researchers and scientists have carried out extensive and in-depth study in the field of software testing using evolutionary algorithms. Ant Colony Optimization (ACO) was proposed by Dorigo et al. [13] in his Ph.D. thesis in 1992. This meta heuristic algorithm draws inspiration from ants searching for their food. It was in the year 2003 that ACO was first put forward by McMinn and Holcombe [14] for generating test cases. Gradually, researchers analyzed this possibility and came out with tools for software testing based on Ant Colony Optimization. Recently, Sharma et al. [1] developed ESCov, a tool based on ACO generation state transition test sequence and achieved maximum coverage with least possible redundancy at the same time.

Suri and Singhal [2] proposed ACO as a promising technique for test case prioritization. The tool developed by Suri and Singhal [2] reduced the number of test cases by 62.5%. Singh et al. [3] have also applied ACO on test case prioritization and their research has yielded successful results. Shunkun Yang, Tianlong Man, and Jiaqi Xu [4] introduced improved ACO for software test cases generation and compared the performance and efficiency of the improved version with Random Algorithms and Genetic Algorithms. Kaur and Goyal [5] put forward Genetic Algorithm for prioritization using code coverage. Ahlam Ansari et. al [6] also proposed an optimized ACO technique for reducing resource utilization and uncovering maximum faults at the same time. Zheng Li, Mark Harman, and Robert M.

Revised Manuscript Received on March 16, 2020.
***** Correspondence Author
Tina Sachdeva, Department of Computer Science, Shaheed Rajguru College of Applied Sciences for Women, University of Delhi, Delhi, India. E-mail: sachdeva_tina@yahoo.com

Hierons[7] studied five search techniques: two meta–heuristic search techniques together with three greedy algorithms and the results obtained indicate that Genetic Algorithms are a promising approach for prioritization.

Suri and Mangal [8] proposed a hybrid approach using Bee Colony Optimization and Genetic Algorithm for test suite minimization and implemented the proposed approach through a tool named HBG_TCS. Thus, above works clearly indicate the popularity of Swarm Optimization and Genetic Algorithms for prioritizing test cases.

III. ANT COLONY OPTIMIZATION

Ant Colony Optimization is a population based meta heuristic algorithm that uses artificial ants for finding solutions to discrete and continuous problems. It simulates the behavior of real ants. In real life, ants are blind and communicate with one another using antennas and pheromone liquid. While hunting for food, all ants traverse a particular path and leave behind a chemical substance known as 'pheromone'. The rest of the ants follow the path by smelling the pheromone left behind. Hence, the optimal path is decided through teamwork involving pheromone deposition and evaporation, and are defined as follows:

i. **Pheromone Deposition**: In this, all ants add pheromone liquid by dropping it on the path they traverse.
ii. **Pheromone Trail Evaporation**: This involves reducing the degree of pheromone on all paths with due passage of time.

The generic algorithm for ACO is:

Initialize pheromone trail, number of ants, test cases and number of iterations.

populate ants
while (condition= =true)
{
for (l=1; l<=no_ants; ++l)
for each ant, evaluate the fitness function f (k) and find probabilistically the edge to be travelled by it
$p^k_{xy} = ((r^{\alpha}_{xy})\ (\eta^{\beta}_{xy})\ ) / \sum_z (r^{\alpha}_{xz})\ (\eta^{\beta}_{xz})$
where z ε permissible

update pheromone trail using the formula
$r_{xy} \leftarrow (1- \theta)\ r_{xy} + \sum \Delta\ r^k_{xy}$
end for

find the best path
}

In the previous algorithm:
i. p^k_{xy} represents probability of transition from x to y
ii. r_{xy} represents the quantity of pheromone deposited while going from x to y
iii. α controls the influence of $\mathbf{r_{xy}}$
iv. η_{xy} represents the desirability of state transition xy as computed by the heuristic function
v. β controls the influence of $\boldsymbol{\eta_{xy}}$
vi. θ pheromone evaporation coefficient
vii. $\Delta\ r^k_{xy}$ represents the quantity of pheromone left by k^{th} ant

IV. BEE COLONY OPTIMIZATION(BCO) TECHNIQUE

Bee Colony Optimization is a nature-inspired swarm-based algorithm that mimics the strategy of foraging honey bees. This algorithm was proposed by Karaboga in 2005[9]. Honey bees have the capability to extend in multiple directions over long distances. This enables them to explore more patches of flowers with sufficient nectar. During foraging, scout bees are sent for searching good flower patches. On return, the scout bees perform 'waggle dance' through which they communicate information to the colony. The waggle dance communicates three parameters regarding a flower patch: the direction of the newly discovered patch, its shortest distance from the hive and the value of its fitness function (quality) [10]. This information is used by other bees in the colony to precisely locate these flower patches as the scout bee leads the follower bees to them. This ensures that the other bees of the colony collect nectar in an efficient and a quick manner. The two main steps of BCO are [11]:

i. **Foraging**: The solution generation phase.
ii. **Waggle Dance**: The information exchange phase which checks the quality of the existing solutions and directs the generation to the new ones.

The generic algorithm for BCO is:

```
for (l=1 ;l<=s_bees; l++)
    bee[l]=create_scout()
    good_site[l]=get_soln(bee[l])
do{
        recruitment();

        for(int m=1;m<no_bees;m++)
            good_site[m]=local_search(good_site[m])
            good_site[m]=leave_site(good_site[m])

    good_site[m]=shrink_solnspace(good_site[m])
        for(int n=no_bees;n<no_s_bees;++n)
            good _site[n]=global_search(good_site[n])
  } while(condition==TRUE)
```

In the previous algorithm:
i. **create_scout ()**: Inserts scout bees into the search space randomly.
ii. **get_soln ()**: The scout bees evaluate the fitness of the flower patch on which they land.
iii. **recruitment ()**: After the scout bees performs waggle dance, the onlooker bees are employed to search neighborhood patches.
iv. **local_search ()**: Explores the neighborhood flowers for better fitness value.
v. **leave_site()**: If a flower patch with better fitness value is obtained, the previous flower is discarded.
vi. **shrink_solnspace ()**: This function reduces the size of solution space by discarding patches with a lower fitness value.

vii. **global_search ()**: Determines the optimized output out of all local search results.

V. GENETIC ALGORITHMS(GA)

Genetic Algorithm is an adaptive meta heuristic search procedure based on the concept of biological selection and evolution. It enhances a population of separate solutions in a recursive manner. At each step, it randomly chooses individuals from the present population and uses them as parents to produce off springs for the next generation. Thus, the population moves towards an optimized result over consecutive generations.

The three main underlying processes in this are:

i. **Selection**: Examines the fitness of an individual allowing the fit ones to pass on their genes to the next generation.

ii. **Crossover**: Interchanging an allele of an individual with another from a different individual. The formula mentioned below is a proposed implementation of crossover [5]:
off spring1 = cr*p1 + (1-cr) *p2 (cr: chromosome)
off spring2 = (1-cr) * p1 + cr*p2

iii. **Mutation**: Allele of genes is randomly replaced by another to produce a new individual. The primary purpose of mutation phase is to maintain diversity in the population and avoid early untimely convergence.

```
// Initialize generation
  x = 0
GP =population of randomly-generated organisms;
Calculate fitness value for each individual 'x'
belonging to the generation 'GP_x'
do {
      create_Nextgeneration();
      //Selection
      best_pop=elit_rate*p_size
      send the best_pop to the next generation
      GP_x+1 rem _pop=p_size- best_pop;
      //Crossover
         for (x=1; x<=rem_pop/2; x++)
                    randomly select 2 organisms Q1 and Q2
                    from the remaining population
                    crossover (Q1, Q2) //subparts of 2 parents
                    //are swapped to produce 2 off springs
         end for
   //Mutation
for(y=1;y<=no_crossover;y++)
     select an individual from the crossover population
     and now mutate its each bit using μ
end for
best_soln=eval_pop();//returns the best solution by
//comparing the fitness values of each individual
 }while fitness of fittest individual in GPy is not up to the
mark
```

VI. RESULTS OF TEST SUIT MINIMIZATION USING GENETIC ALGORITHMS AND ANT COLONY OPTIMIZATION

In this section, first Genetic Algorithms and then Ant Colony Optimization implementation is used for the purpose of test suite prioritization.

In implementation using Genetic Algorithms, single point crossover operation was used and the crossover probability was computed as a scaled pseudorandom number R8 between 0 and 1. Next, the crossover point was taken as a pseudorandom number I4 falling somewhere between 0 and number of variables in each individual. In the operation of mutation, the variable to undergo mutation was picked up in a random manner and was replaced with another random value between the upper and lower bound of that variable. Crossover and mutation were performed provided their probability of execution was less than the initial pre-set probability. A user defined function time_count() computed the time of execution till specified number of generations were reached. Another user defined function compute () evaluated the fitness value of each individual which was taken as the objective function

During the initial run of this implementation, initial count of individuals in population, maximum number of generations, probability of crossover and probability of mutation in the program were chosen. These variables can be easily reinitialized to any value suited to the problem under consideration.

To determine potential effectiveness of this implementation, a case study of a GCD program as shown in Fig.1 was carried out.

```
1.  void main ( int x, int y)  { int z;
2.  if( y > x)  {
3.  z = x;
4.  x = y;
5.  y =z;
6.  z = x % y;
7.  while ( z !=  0) {
8.  x = y;
9.  y = z;
10. z = x  %  y;
11. }
12. return y;
13. }
```

Fig.1: Program to find GCD of two numbers

Fig.1. shows the GCD program (assumes smaller of the two numbers is inputted first) that accepts two integer parameters x and y and computes their greatest common divisor or highest common factor and outputs it as z.

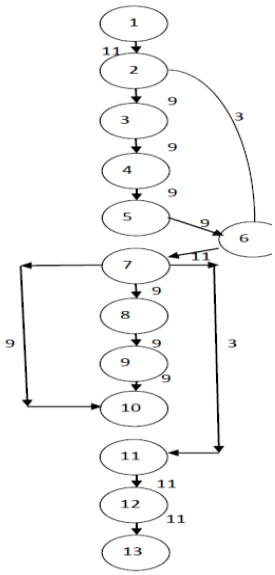

Fig 2: Control flow graph of the GCD program

The simple independent paths from Fig.2 can be easily inferred as:

1. **P1**: 1-2-6-7-11-12-13
2. **P2**:1-2-6-7-8-9-10-7-11-12-13
3. **P3**:1-2-3-4-5-6-7-11-12-13
4. **P4**:1-2-3-4-5-6-7-8-9-10-7-11-12-13

Fitness function for the problem based on path dependency is given as:

$$f(x)=\sum W_i \text{ for all } i=0 \text{ to } n,$$

where W_i denotes the weights assigned to the respective paths.

The implementation was executed for the GCD program of Fig.1 and it was also executed with randomly generated test data. The results summarized in Table 1 show that for the same parameter settings of input, random testing took execution time which was on average five times or more when compared to implementation using Genetic Algorithms to reach the same fitness value for which the associated values of variables can serve as capable enough test data for the purpose of error detection.

Table I: Time taken by Genetic Algorithm based implementation and random testing for GCD program

Testing Number	Time taken by GA implementation (in msec)	Time taken by Random Testing (in msec)
1	2.1	11.5
2	1.06	8.8
3	3.4	13.99
4	2.86	16.07
5	1.1	8
6	3.1	19.66
7	2.8	12.9

Next, for the same GCD program, Ant Colony Optimization was used.

Table II: Time taken by Ant Colony Optimization based implementation for GCD program

Testing Number	Time taken by ACO implementation (in msec)
1	2
2	1.1
3	2.9
4	2.4
5	1.0
6	2.98
7	2.5

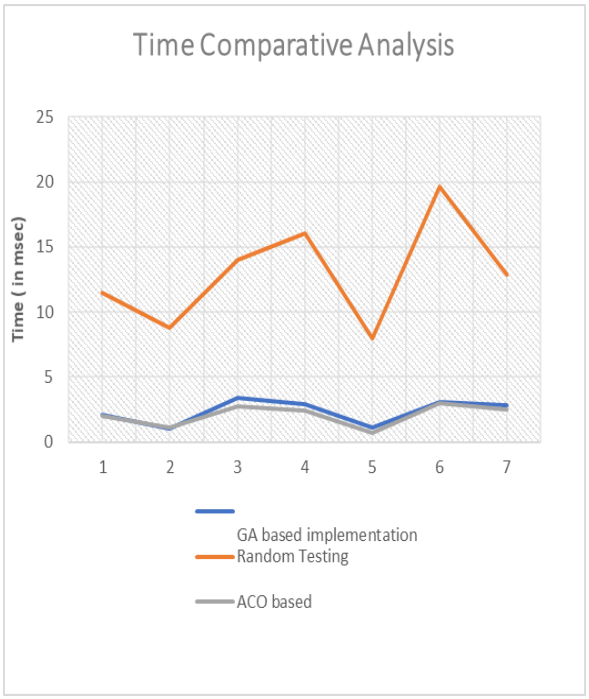

Fig 3: Execution time comparison of three techniques

VII. DISCUSSION OF COMPARISON OF SWARM INTELLIGENCE AND GENETIC ALGORITHMS

Experimental results of the previous section show that Genetic Algorithms and Ant Colony Optimization are far better in terms of execution time for test case prioritization as compared to naive prioritization techniques. Moreover, this execution time for Genetic Algorithm based implementation is quite close to Ant Colony Optimization based implementation with former performing slightly better. However, random testing approach takes time which is almost five times or even more.

Further, quite a few studies have been done on the application of Swarm Optimization and GA in generating prioritized test suites. Research scholars have devised tools that employ improved variations of ACO, BCO and GA.

Shunkun Yang, Tianlong Man, and Jiaqi Xu[4] have proposed enhanced versions of Ant colony Optimization namely- : better local pheromone update logic for ant colony optimization, better pheromone volatilization coefficient for ant colony optimization (IPVACO), and better global path pheromone update strategy for ant colony optimization (IGPACO).

Also, they have suggested a comprehensive Improved Ant Colony Optimization (ACIACO) which is based on the above mentioned three techniques. The authors then came out with a comparison between the proposed technique Random Algorithm (RND) and Genetic Algorithm in terms of both efficiency and coverage. Their results showed that performance of GA, ILPACO and IGPACO was almost comparable in terms of statement coverage. The coverage of test cases by IPVACO was better than ILPACO. IGPACO picked up best ant to update pheromone and had a good astringency. The minimum number of iterations, where branch coverage achieved 100%, was much smaller than IPVACO and IGPACO. ACIACO, which is based on all the above three methods, enhanced the search efficiency, showed promising level of coverage and greatly reduced the number of iterations. Bharti Suri et al. [2] have designed a tool named ACO_TCSP which prioritizes the test suite to ensure total fault detection at minimized cost. The results are promising as the tool reduces the size of the test suite by 25%. Moreover; there is no requirement to travel through all paths for detecting all faults in a specified time. Mitras and Adeeba Khaboo [12] approach unites GA and Continuous Ant Colony Optimization (CACO) to produce optimized solutions. The hybrid algorithm uses Continuous Ant Colony Algorithm as a mutation of Genetic Algorithm. The authors evaluate the efficiency of the suggested approach using a set of standard functions.

Suri and Mangal [8] have also suggested a novel technique of combing BCO with GA. In their study, they develop a tool named HBG_TCS that implements the proposed approach. This hybrid approach comes out be much faster than the native ACO technique.

VIII. CONCLUSION AND FUTURE WORK

From the above discussion, it is evident that Swarm Optimization Techniques and Genetic Algorithms prove to be more beneficial than the traditional prioritization methods being used earlier. Also, it is evident that hybridized algorithms perform better than the stand-alone Swarm Intelligence Techniques. ACO and GA, both have some inherent disadvantages. Because of these, it is preferable to use Genetic Algorithm, Artificial Bee Colony, or other heuristic algorithms in combination with Ant Colony Optimization so that two algorithms mutually complement each other. Such comprehensive techniques will enhance the capability of the software test suite generated effectively.

Also, much work has not been done in test case prioritization using artificial intelligence techniques namely, Intelligent Water Drops, Gravitational Search Algorithm, Simulated Annealing, Stochastic Diffusion Search etc. It would be worthwhile to explore test case prioritization using these techniques too.

REFERENCES

1. B. Sharma, I. Girdhar, M. Taneja, P. Basia, S. Vadla, and P. R. Srivastava, “Software coverage: a testing approach through ant colony optimization,” in Proceedings of the 2nd International Conference on Swarm, Evolutionary, and Memetic Computing (SEMCCO '11), pp. 618–625, 2011.
2. B. Suri and S. Singhal, “Implementing Ant colony optimization for test case selection and prioritization,” International Journal on Computer Science and Engineering, vol. 3, no. 5, pp. 1924–1932, 2011.
3. Y. Singh, A. Kaur, and B. Suri, “Test case prioritization using ant colony optimization,” ACM SIGSOFT Software Engineering Notes, vol. 35, no. 4, pp. 1–7, 2010.
4. Shunkun Yang, Tianlong Man, and JiaqiXu, “Improved Ant Algorithms for Software Testing Cases Generation,” The Scientific World Journal, vol. 2014, Article ID 392309, 9 pages, 2014. doi:10.1155/2014/392309
5. A. Kaur, and S. Goyal, “A Genetic Algorithm For Regression Test Case Prioritization Using Code Coverage”, International Journal on Computer Science and Engineering, vol 3, no 5, pp. 1839-1847, 2011.
6. A. Ansari, A. Khan, A. Khan, and K. Mukadam, “Optimized Regression Test using Test Case Prioritization”, Proceedings of International Conference on Communication,Virtualization (ICCCV), Volume 79, Pages152-160, 2016
7. Zheng Li,Mark Harman, and Robert M.Hierons,”Search Algorithms for Regression Test Case Priortization”, IEEE Transactions on Software Engineering, Vol.33, No.4,pp.225-237,2007
8. Bharti Suri, and Isha Mangal, “Analyzing Test Case Selection using Proposed Hybrid Technique based on BCO and Genetic Algorithm and a comparison with ACO”,International Journal of Advanced Research in Computer Science and Software Engineering, Volume 2, Issue 4, April 2012.
9. D. Karaboga,B. Basturk, “Artificial Bee Colony(ABC) Optimization Algorithm for Solving Constraint Optimization Problems”, Advances in Soft Computing: Foundations of Fuzzy Logic and Soft Computing, Vol:4529/2007, pp: 789-798,Springer- Verlag,2007,IFSA 2007
10. Camazine S,Denebourg J,Franks NR,Sneyd J, Theraula G and Bonabeau E.Self-Organization in Biological Systems.Princeton:Princeton University Press 2003
11. Tatjana DAVIDOVIC, Dusan TEODOROVIC, and Milica SELMIC, “Bee Colony Optimization Part 1:The Algorithm Overview”,Yugoslav Journal of Operations Research 25(2015), Number 1,33-56
12. Ban A. Mitras and AdeebaKH.Aboo,”Hybrid of Genetic Algorithm and Continous Ant Colony Optimization for Optimum Solution”, International Journal of Computer Networks and Communications Security, VOL.2, NO.1, jANUARY 2014,pp 1-6
13. M.Dorigo, V.Maniezzo, and A.Colorni,”Ant system: optimization by a colony of cooperating agents,”IEEE Transactions on Systems,Man and Cybernetics B:Cybernetics, vol.26,no 1,pp29-41,1996
14. P.McMinn and M.Holcombe, “The state problem for evolutionary testing,” in proceedings of the Genetic and evolutionary Computation Conference(GECCO’03),pp.2488-2498,Springer,Chicago,111,USA,July 2003.

AUTHORS PROFILE

Tina Sachdeva, is currently working as an Assistant Professor in the Department of Computer Science of Shaheed Rajguru College of Applied Sciences for Women, University of Delhi with about 12 years of teaching experience. She has authored several National and International research publications.