



Estimating Software Reliability Using Ant Colony Optimization Technique with Salesman Problem for Software Process

¹D. Hema Latha , ²Prof. P. Premchand

¹Research Scholar, Dept of Computer Science, Rayalaseema University, Kurnool, Andhra Pradesh, India

²Professor, Dean, Faculty of Informatics, Dept of Computer Science and Engineering, UCE, Osmania University, Hyderabad, TS, India

ABSTRACT

Software reliability means it is a failure free operation of software for a specific period of time under specified environment. Software reliability is defined as the probability with which the software will operate without any failure for a specific period of time in a specified environment. It is one of the important software quality features. Software reliability, when estimated in early phases of software development life cycle, saves lot of money and time as it prevents spending huge amount of money on fixing of defects in the software after it has been deployed to the client. Software reliability prediction is very challenging in starting phases of life cycle model. Software reliability estimation has thus become an important research area as every organization aims to produce reliable software, with good quality and error or defect free software. There are many software reliability growth models that are used to assess or predict the reliability of the software. These models help in developing robust and fault tolerant systems.

In the past few years many software reliability models have been proposed for assessing reliability of software but developing accurate reliability prediction models is difficult due to the recurrent or frequent changes in data in the domain of software engineering. As a result, the software reliability prediction models built on one dataset show a significant decrease in their accuracy when they are used with new data. The objective of this paper is to introduce a new approach that optimizes the accuracy of software reliability predictive models when used with raw data. Ant Colony Optimization Technique (ACOT) is proposed to predict software reliability based on data collected from literature. An ant colony system by combining with Travelling Sales Problem (TSP) algorithm has been used, which has been changed by implementing different algorithms and extra functionality, in an attempt to achieve better software reliability results with new data for software process.

The intellectual behavior of the ant colony framework by means of a colony of cooperating

artificial ants are resulting in very promising results. The method is validated with real dataset using Mean Time to Failure (MTTF) and Mean Time Between Failure (MTBF).

Keywords: Software Reliability, Bio-inspired Computing, Ant Colony Optimization (ACO) technique, Travelling Salesman Problem (TSP).

1 INTRODUCTION

As the past decades have seen the computerization of all the functionalities in all the fields turn out to be supplementary multifaceted and therefore, there is a constant demand for discovering innovative well organized methodologies to software development and preservation. There is a prerequisite of the enormous scope of effort, time and currency to arrange and build up any feasible software apart from the human resource and their organization. For outstanding rising competition, today's profitable conditions have become very dynamic. Corporate industries require proceeding extremely fast to unstable needs of the market. Hence, software engineering which emphasizes with all these domains has become an individual study from researchers. The software crisis is defined as mismatch between what the software can deliver and the capacities of computer systems, as well as the expectations of their users and where software problems cause the system tasks to be delayed, expensive, and/or not amenable to the user's desires. The software can be developed to meet the various stages of reliability, security,

portability, usability, effective cost and response time.

Developing awfully trustworthy software from the user's perspective is a demanding profession for all software engineers. However, Software Reliability [1], [2], [3] is a significant aspect influencing system reliability. The following four practical aspects which are related to achieving reliable software systems and these aspects furthermore are treated as four fault Lifecycle techniques:

1) Fault avoidance: to avoid, by building, error existence. 2) Fault elimination: to identify, by confirmation and proof, the presence of faults and eliminate them. 3) Fault tolerance: to specify, by redundancy, facility conforming to the requirement in spite of faults having rising. 4) Fault/failure Predicting to estimate: by the assessment, the occurrence of faults and consequences of failures. Quality [10], [11] is an important feature of reliability.

Software reliability is the probability that software will not cause failure of a system for a particular point in time underneath particular circumstances. The probability is a function of the inputs to and use of the system as well as a function of the existence of faults in the software. According to ANSI, Software Reliability [13], [14] is defined as: "the probability of failure free software operation for a particular period of time in a particular atmosphere". Software reliability evaluation is significant to evaluate and forecast the trustworthiness and performance of software systems. Reliability representation is a crucial ingredient of the reliability evaluation procedure and it also validate whether a product meets up its reliability objective and is ready to distribute. The fundamental intention of most of software reliability models is making them to understand distinctiveness and reasons to fail software, and try to enumerate software reliability. The current

article lay emphasis on about a bio inspired computing technique Swarm Intelligence known as the Ant Colony Optimization Technique to predict software reliability. The anticipated method is employed into a TSP problem with software failure datasets to predict software reliability and the results of our approach are reported. And, thus, the focus of the discussion to be presented here is an ACO for discrete optimization that has been used to predict software reliability using the Travelling Sales Person Problem where failure data is given as input and the result is calculated through Mean Time to Failure (MTTF) and Mean Time Between Failure (MTBF) to predict the reliability.

2 METHODOLOGY

A. Bio Inspired Computing

Natural computing [22] is a term presented to comprise three classes of methods: (1) those that take motivation from nature for the development of novel problem-solving techniques; (2) those that are constructed with the use of computers to synthesize natural facts; and (3) those that employ natural resources (e.g., molecules) to compute. The main areas of research that comprise these three branches are the artificial neural networks,[4], [5], [6], [16], [17], evolutionary algorithms, swarm intelligence [20], artificial immune systems, fractal geometry, artificial life, DNA computing, and quantum computing, among others. Bio-inspired Computing is the combination of computational aptitude and collective intelligence. These computational approaches are used to resolve multifaceted problems, and developed after design principles confronted in natural or biological systems, and tend to be adaptive, responsive, and distributed. The aim of bio-inspired computing [7] is to develop computational tools with enhanced strength, scalability,

flexibility and which can interact more efficiently with humans. It can offer biologists, for example, with an IT-oriented concept for looking at how cells compute or process information, or help computer scientists build algorithms based on natural systems, such as evolutionary and genetic algorithms.

Biocomputing [23] has the potential to be a very powerful tool. The association of bio-inspired and soft computing techniques [18], [19] are artificial neural networks [8], [9], [10], evolutionary algorithms, swarm intelligence, artificial immune systems [15], fractal geometry, artificial life, DNA computing and quantum computing.

B. Ant Colony Optimization Technique

Ant Colony [24-27] is one of the techniques of bio inspired computing. The main concept of this technique is that the self-organizing rules which allow the highly synchronized behavior of real ants can be utilized to manage populations of artificial agents that cooperate to solve computational problems. Various distinctive attributes of the behavior of ant colonies have inspired different kinds of ant algorithms. Examples are foraging, distribution of labor, issue sorting, and cooperative transport. Ants coordinate their activities via stigmergy, a form of implicit interaction mediated by changes in the environment. For example, a foraging ant deposit a chemical on the ground which raises the probability that other ant will follow the same path. Biologists have presented that many colony-level behaviors witnessed in social insects can be described through relatively simple models in which only stigmergic communication is present. In other words, biologists have shown that it is often sufficient to consider stigmergic, indirect communication to explain how social insects can attain self-organization. The notion

behind ant algorithms is to use a form of artificial stigmergy to coordinate societies of artificial agents. One of the most effective examples of ant algorithms is known as “ant colony optimization”, or ACO. ACO is motivated by the foraging behavior of ant colonies, and targets discrete optimization problems. The ants may deposit a pheromone on the ground while returning back to their nests. The ants follow with high probability pheromone trails their sense on the ground.

Each Ant evaluates the next move to another vertex based on Gambardella et al., [28, 29],

$$p_{ij}^k = \begin{cases} \frac{(\tau_{ij})^\alpha (\eta_{ij})^\beta}{\sum (\tau_{ij})^\alpha (\eta_{ij})^\beta} & \text{if } j \in \text{allowed } k \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

p_{ij}^k is the probability for a worker K to move to vertex “ ij ”

τ_{ij} is the amount of pheromone deposited on edge to “ ij ”

η is the inverted distance, describes how fast ants select their path.

The tour cost of each ant is given by d_{ij} the tour cost from the city i to city j (edge weight) is calculated and hence the shortest path is found. This is applied to the Travelling Sales Person Problem and optimized solutions are obtained using

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \quad (2)$$

The amount of pheromone deposited by each ant is given by

$$\tau_{ij}(t+1) = \rho \tau_{ij}(t) + \Delta \tau_{ij} \quad (3)$$

The value of $\Delta \tau_{ij}$ is calculated using

$$\Delta \tau_{ij}(t) = \sum_{k=1}^m \Delta \tau_{ij}^k(t) \quad (4)$$

$\Delta \tau_{i,j}^k$ is calculated using

$$\Delta \tau_{ij}^k = \begin{cases} \frac{Q}{L_k} & \text{if } (i, j) \in \text{bestTour} \\ 0 & \text{otherwise} \end{cases}$$

C. Algorithm

The ACO algorithm [30] which has been proposed based on the study that real ants are skilled in finding the shortest path from a food source to the nest without using visual signals. From the originating point the ants start the tour selecting randomly any path. The ACO algorithm is as follows:

1. Set the initial parameters.
2. Initialize pheromone trails.
3. Calculate the maximum specific ways in which the ants can travel.
4. Loop //iteration
5. Each ant is positioned at a given node randomly selecting the node according to some distribution strategy (each node has at least one ant)
6. For $k=1$ to m do //steps in a loop
7. The first step: move each ant in a different route
8. Repeat //till all the nodes are visited
9. Select node j to be visited next // the next node must not be an already visited node
10. Apply local updating rule
11. Until ant k has completed a tour
12. End for
13. Apply sub tour that is sub Local search // to improve tour
14. Apply global updating rule
15. Compute entropy value of current pheromone trails

16. Update the heuristic parameter
17. Until End_condition
18. End

The flow chart for Ant Colony Optimization (ACO) algorithm with travelling salesman problem is shown in fig(1)

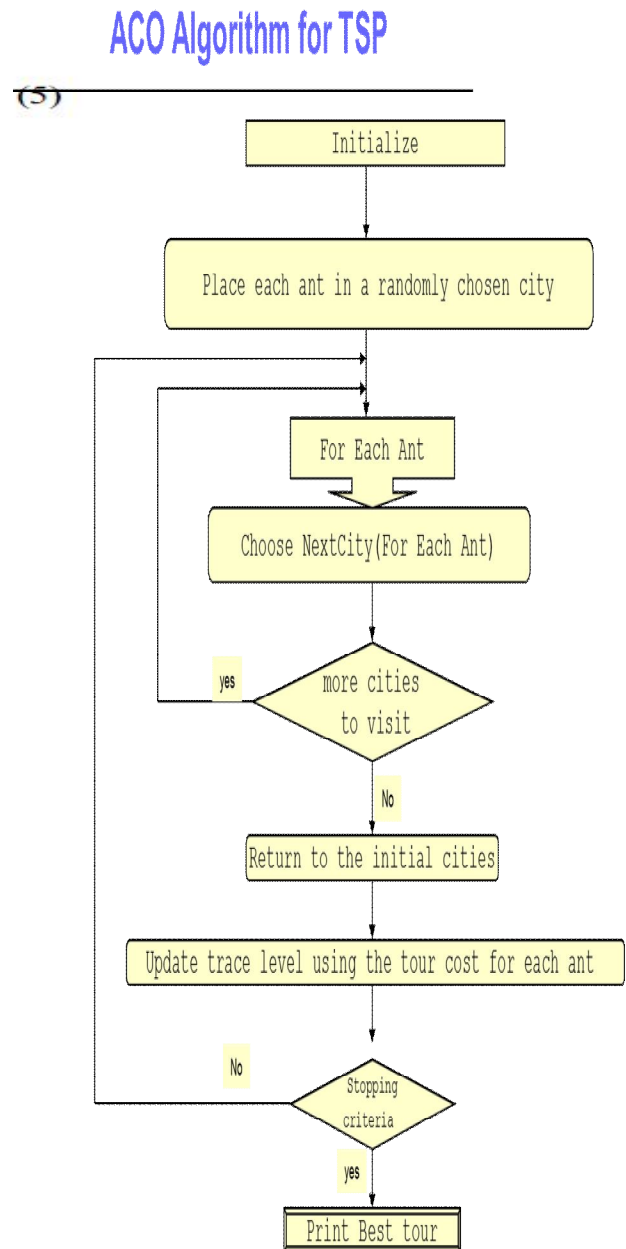


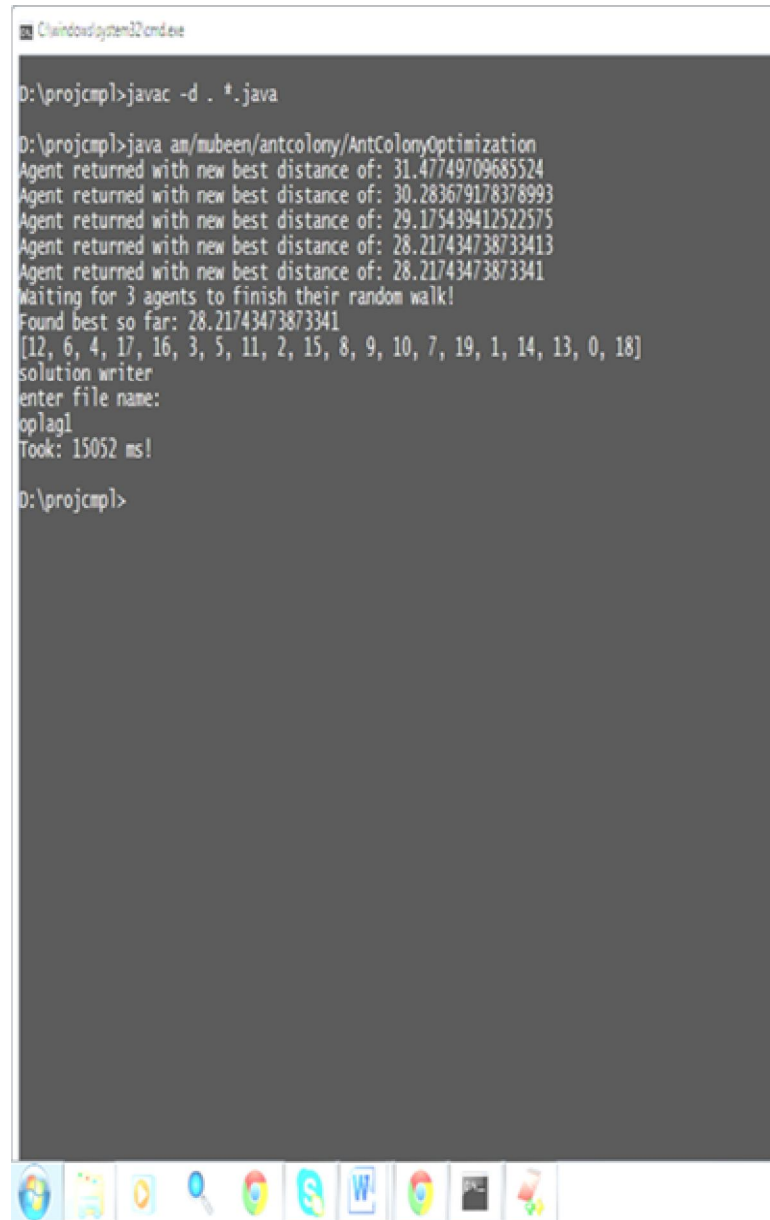
Figure. 1. Flow chart of the ACO algorithm

3 IMPLEMENTATION RESULTS

In this experiment, time series forecasting model is employed to predict software reliability which has only one dependent variable and no explanatory variables. In this paper, the software failure data obtained from Musa [21] data sets is employed in this study. It is used to demonstrate the forecasting performance of Ant colony optimization techniques. The data contains 101 observations of the pair (t, Yt) pertaining to software failure. Here Yt represents the time to failure of the software after the t th modification has been made. Five data sets are created lag # 1,2,3,4 and 5 in view of the foregoing discussion on generating lagged data sets out of a time series.

Implementation results are shown in the following screens:

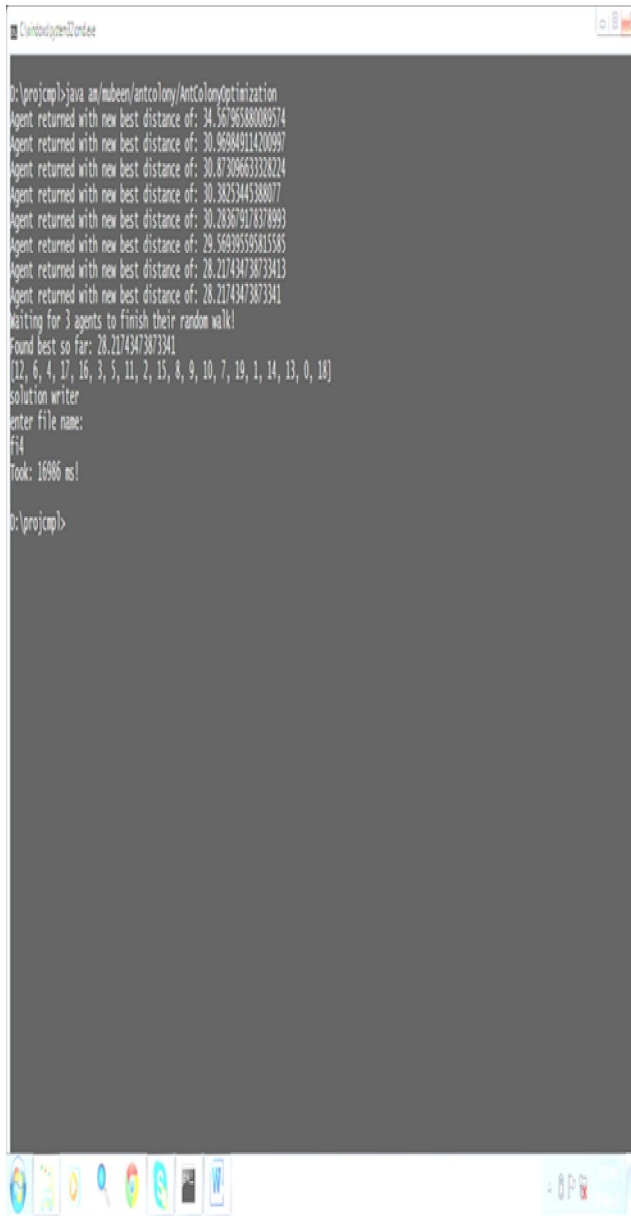
Best or optimized distance travelled by artificial ants from source to destination is 28.21743473873341 and average time taken to travel is 15052 ms, shown in figure (2).



```
Windows\system32\cmd.exe
D:\projcmp>javac -d . *.java
D:\projcmp>java am/mubeen/antcolony/AntColonyOptimization
Agent returned with new best distance of: 31.47749709685524
Agent returned with new best distance of: 30.283679178378993
Agent returned with new best distance of: 29.175439412522575
Agent returned with new best distance of: 28.217434738733413
Agent returned with new best distance of: 28.21743473873341
Waiting for 3 agents to finish their random walk!
Found best so far: 28.21743473873341
[12, 6, 4, 17, 16, 3, 5, 11, 2, 15, 8, 9, 10, 7, 19, 1, 14, 13, 0, 18]
solution writer
enter file name:
oplag1
Took: 15052 ms!
D:\projcmp>
```

Figure.2. Time taken and distance travelled for artificial ants to reach destination - screenshot 1

Best or optimized distance travelled by artificial ants from source to destination is 28.21743473873341 and average time taken to travel is 16986 ms, shown in figure (3).



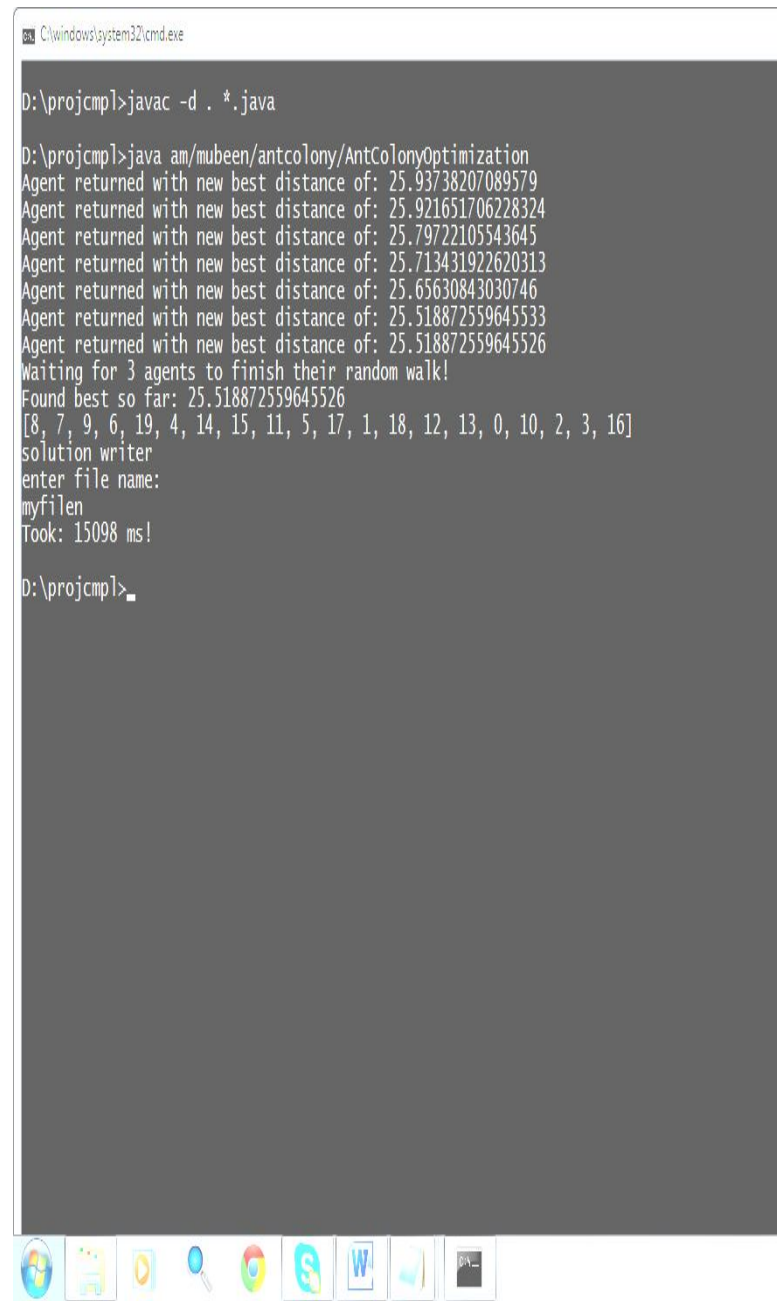
```
D:\projcml>java am/mubeen/antcolony/AntColonyOptimization
Agent returned with new best distance of: 34.567965800089574
Agent returned with new best distance of: 30.969849114200997
Agent returned with new best distance of: 30.873096633328224
Agent returned with new best distance of: 30.382534453880077
Agent returned with new best distance of: 30.283679178378993
Agent returned with new best distance of: 29.569395595825585
Agent returned with new best distance of: 28.217434738733413
Agent returned with new best distance of: 28.21743473873341
Waiting for 3 agents to finish their random walk!
Found best so far: 28.21743473873341
[12, 6, 4, 17, 16, 3, 5, 11, 2, 15, 8, 9, 10, 7, 19, 1, 14, 13, 0, 18]
solution writer
enter file name:
ff4
Took: 16986 ms!

D:\projcml>
```

Figure.3. Travelling time for artificial ants from source to destination – screen shot 2

Best or optimized distance travelled by artificial ants from source to destination is 25.518872559645526 and average time

taken to travel is 15098 ms, shown in figure (4).



```
C:\windows\system32\cmd.exe

D:\projcml>javac -d . *.java

D:\projcml>java am/mubeen/antcolony/AntColonyOptimization
Agent returned with new best distance of: 25.93738207089579
Agent returned with new best distance of: 25.921651706228324
Agent returned with new best distance of: 25.79722105543645
Agent returned with new best distance of: 25.713431922620313
Agent returned with new best distance of: 25.65630843030746
Agent returned with new best distance of: 25.518872559645533
Agent returned with new best distance of: 25.518872559645526
Waiting for 3 agents to finish their random walk!
Found best so far: 25.518872559645526
[8, 7, 9, 6, 19, 4, 14, 15, 11, 5, 17, 1, 18, 12, 13, 0, 10, 2, 3, 16]
solution writer
enter file name:
myfilen
Took: 15098 ms!

D:\projcml>_
```

Figure.4. Time taken for artificial ants to reach destination – screen shot 3

Best or optimized distance travelled by artificial ants from source to destination is

28.21743473873341 and average time taken to travel is 19170 ms, shown in figure (5).

```
C:\windows\system32\cmd.exe
D:\projcmp]javac -d . *.java
D:\projcmp]>
D:\projcmp]>java am/mubeen/antcolony/AntColonyOptimization
Agent returned with new best distance of: 37.52603085446347
Agent returned with new best distance of: 30.873096633328228
Agent returned with new best distance of: 29.175439412522575
Agent returned with new best distance of: 28.217434738733413
Agent returned with new best distance of: 28.21743473873341
Waiting for 3 agents to finish their random walk!
Found best so far: 28.21743473873341
[12, 6, 4, 17, 16, 3, 5, 11, 2, 15, 8, 9, 10, 7, 19, 1, 14, 13, 0, 18]
solution writer
enter file name:
laglin
Took: 19170 ms!
D:\projcmp]>
```

Figure. 5. Traversing time for artificial ants from source to destination – screen shot 4

Best or optimized distance travelled by artificial ants from source to destination is 28.21743473873341 and average time taken to travel is 22028 ms, shown in figure (6).

```
C:\windows\system32\cmd.exe
D:\projcmp]javac -d . *.java
D:\projcmp]>
D:\projcmp]>java am/mubeen/antcolony/AntColonyOptimization
Agent returned with new best distance of: 42.48601579355241
Agent returned with new best distance of: 41.532868255487216
Agent returned with new best distance of: 37.71739206707082
Agent returned with new best distance of: 35.649177168681014
Agent returned with new best distance of: 35.60878277493641
Agent returned with new best distance of: 35.6087827749364
Waiting for 3 agents to finish their random walk!
Found best so far: 35.6087827749364
[15, 3, 12, 13, 10, 11, 18, 5, 19, 6, 7, 9, 2, 1, 8, 14, 0, 4, 17, 16]
solution writer
enter file name:
oplag4
Took: 22028 ms!
D:\projcmp]>
```

Figure.6. Time taken for artificial ants to reach destination – screen shot 5

4 CONCLUSION

ACO is a comparatively new metaheuristic concept for resolving tough combinatorial optimization problems. Simulated or Artificial ants realize a random construction heuristic approach which compose a probabilistic judgment. The cumulated search practice is taken into consideration by the adapting the pheromone trail. ACO exhibits great performance when used for “ill-structured” problems like network congestion and routing. When ACO local search is implemented is mandatory to obtain optimistic results.

Ant algorithms fit into a group of Meta heuristics, which are known for range of applications to realistic problems encountered in scientific, business applications and industrial scenarios. A variety of applications depicted in this study focuses on ant algorithms that can be applied to plenty of sensible situations. The algorithms employed in this work are inspired by an observation emphasizing on real ants nature i.e. foraging and searching abilities that can provide good answers to genuine and real time optimization and software reliability solutions. The indirect interaction and the co-operative communication of the simulated ant agents is enthused from their actual living counterpart, exhibiting great elasticity and sensitivity to vibrant problems. The function of these programs and investigational validations are enormously studied owing to their potential to offer most favorable generic solutions to specific complex problems such as imaging problems, local search, compression theory, image mapping and searching databases.

This research paper represents ACO methodology and its implementation. The exploration is still in progress as many of the facets of ACO algorithm are still to be

unrevealed. It is expected that this study stimulates further discussion for better reliability solutions.

5 FUTURE WORK

The prospects of Ant algorithm based applications with reference to the geometric tolerance amalgamation and distribution of the potential and probable regions of exploration.

The investigation of more efficient pheromone models might reduce the need of comprehensive intensification phases and the future must evolve theoretical development of models for experimentation and for creating effective models.

REFERENCES

- [1] R. K. Mohanty, V. Ravi, and M. R. Patra, “Hybrid intelligent Systems for predicting Software reliability,” Elsevier, Applied Soft Computing, vol. 13, No. 1, pp. 189-200, 2013.
<https://doi.org/10.1016/j.asoc.2012.08.015>
- [2] R. K. Mohanty, V. Ravi, and M. R. Patra, “Application of Machine learning Techniques to Predict software reliability,” International Journal of Applied Evolutionary Computation, vol. 1, No.3, pp. 70-86, 2010.
<https://doi.org/10.4018/jaec.2010070104>
- [3] K. Cai, C. Yuan, and M. L. Zhang, “A critical review on software reliability modeling,” Reliability engineering and Systems Safety, vol. 32, pp. 357-371, 1991.
[https://doi.org/10.1016/0951-8320\(91\)90009-V](https://doi.org/10.1016/0951-8320(91)90009-V)
- [4] T. Dohi, Y. Nishio, and S. Osaki, “Optional software release scheduling based on artificial neural networks,” Annals of Software engineering, vol. 8, pp. 167-185, 1999.
<https://doi.org/10.1023/A:1018962910992>
- [5] N. Karunanithi, Y. K. Malaiya, and D. Whitley, “The scaling problem in neural networks for software reliability prediction,” In Proceedings of the Third International IEEE Symposium of Software Reliability Engineering, Los Alamitos, CA, pp. 76- 82, 1992a.
<https://doi.org/10.1109/ISSRE.1992.285856>
- [6] N. Karunanithi, D. Whitley, and Y.K. Malaiya, “Prediction of software reliability using connectionist models,” IEEE Transactions on Software Engineering, vol. 18, pp. 563-574, 1992b.
<https://doi.org/10.1109/32.148475>

- [7] T. M. Khoshgoftaar, D. L. Lanning, and A. S. Pandya, "A neural network modeling for detection of high-risk program," In Proceedings of the Fourth IEEE International Symposium on Software Reliability Engineering, Denver, Colorado, pp. 302-309, 1993.
- [8] T. M. Khoshgoftaar, and P. Rebours, "Noise elimination with partitioning filter for software quality estimation," International Journal of Computer Application in Technology, vol. 27, pp. 246-258, 2003.
<https://doi.org/10.1504/IJCAT.2006.011996>
- [9] T. M. Khoshgoftaar, A.S. Pandya, and H.B. More, "A neural network approach for predicting software development faults," In Proceedings of the third IEEE International Symposium on Software Reliability Engineering, Los Alamitos, CA, pp. 83- 89, 1992.
<https://doi.org/10.1109/ISSRE.1992.285855>
- [10] T. M. Khoshgoftaar, E. B. Allen, and J.P. Hudepohl, S.J. Aud, "Application of neural networks to software quality modeling of a very large telecommunications system," IEEE Transactions on Neural Networks, vol. 8, No. 4, pp. 902-909, 1997.
<https://doi.org/10.1109/72.595888>
- [11] T. M. Khoshgoftaar, E.B. Allen, W. D. Jones, and J. P. Hudepohl, "Classification –Tree models of software quality over multiple releases," IEEE Transactions on Reliability, vol. 49, No. 1, pp. 4-11, 2000.
<https://doi.org/10.1109/24.855532>
- [12] J. R. Koza, "Genetic Programming: On the Programming of Computers by Means of Natural Selection". Cambridge, MA: The MIT Press, 1992.
- [13] J. D. Musa, Iannino, A., and K. Okumoto, "Software Reliability, Measurement, Prediction and Application," McGraw-Hill, New York, 1987.
- [14] J. D. Musa, "Software reliability data," IEEE Computer Society- Repository, 1979.
- [15] N. Karunanithi, D. Whitley, and Y.K. Malaiya, "Prediction of software reliability using neural networks," International Symposium on Software Reliability, pp. 124-130, 1991.
<https://doi.org/10.1109/ISSRE.1991.145366>
- [16] T.M. Khoshgoftaar, and R.M. Szabo, "Predicting software quality, during testing using neural network models: A comparative study," International Journal of Reliability, Quality and Safety Engineering, vol. 1, pp. 303-319, 1994.
<https://doi.org/10.1142/S0218539394000222>
- [17] L. Tian, and A. Noore, "Evolutionary neural network modeling for software cumulative failure time prediction," Reliability Engineering and System Safety, vol. 87, pp. 45-51, 2005b.
<https://doi.org/10.1016/j.ress.2004.03.028>
- [18] N. Rajkiran, and V. Ravi. "Software Reliability prediction by soft computing technique," The Journal of Systems and Software, vol. 81, No.4, pp. 576-583, 2007.
<https://doi.org/10.1016/j.jss.2007.05.005>
- [19] N. Rajkiran, and V. Ravi, "Software Reliability prediction using wavelet Neural Networks," International Conference on Computational Intelligence and Multimedia Application (ICCIMA, 2007), pp. 195-197, 2007
- [20] V. Ravi, N. J. Chauhan, and N. RajKiran., "Software reliability prediction using intelligent techniques: Application to operational risk prediction in Firms," International Journal of Computational Intelligence and Applications, vol.8, No. 2, pp. 181-194, 2009.
<https://doi.org/10.1142/S1469026809002588>
- [21] E. Bonabeau, M. Dorigo, and G. Théraulaz, "Inspiration for optimization from social insect behavior," Nature, pp. 39–42, 2000.
<https://doi.org/10.1038/35017500>
- [22] A. Coloni, M. Dorigo, and V. Maniezzo, "Ant system: Optimization by a colony of cooperating agent," IEEE Trans. Systems Man and Cybernetics-Part B: Cybernetics, vol. 26, No.1, pp. 29-41, 1996.
- [23] M. Dorigo and G. Di Caro, "The Ant Colony Optimization Meta-Heuristic," In D. Corne, M. Dorigo and F. Glover, editors, New Ideas in Optimization, McGraw-Hill, pp. 11-32, 1999.
- [24] M. Dorigo, and L. M. Gambardella, "Ant colonies for the traveling salesman problem", BioSystems 43, pp. 73–81, 1997.
[https://doi.org/10.1016/S0303-2647\(97\)01708-5](https://doi.org/10.1016/S0303-2647(97)01708-5)
- [25] M. Dorigo, and L. M. Gambardella, "Ant Colony System: A cooperative learning approach to the traveling salesman problem," IEEE Transactions on Evolutionary Computation, vol. 1, No.1, pp.53–66, 1997.
<https://doi.org/10.1109/4235.585892>
- [26] M. Dorigo, V. Maniezzo, and A. Coloni, "The Ant System: An autocatalytic optimizing process," Technical Report 91-016 Revised, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991.
- [27] L. M. Gambardella, E. D. Taillard, and M. Dorigo, "Ant colonies for the quadratic assignment problem," Journal of the Operational Research Society, vol.50, No.2, pp.167–176, 1999.
<https://doi.org/10.1057/palgrave.jors.2600676>
- [28] V. Maniezzo, and A. Coloni, "The Ant System applied to the quadratic assignment problem," IEEE Transactions on Data and Knowledge Engineering, Vol.11, No. 5, pp. 769– 778, 1999.
<https://doi.org/10.1109/69.806935>

[29] L. M. Gambardella, E. D. Taillard, and G. Agazzi, "MACSVRPTW: A multiple ant colony system for vehicle routing problems with time windows," In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pp. 63–76. Hill, London, UK, 1999.

[30] R. Poli, and W.B. Langdon, J.R. Koza, "A field guide to Genetic Programming," ISBN: 978-1-4092-0073-4, publisher- Lulu.com , United Kingdom, 2008.