International Journal of Advanced Technology in Engineering and Science
Vol. No. 12, Issue No. 07, July 2024
www.ijates.com
ijates
ISSN 2348 - 7550

Enhance Software reliability approach power-law testing effort function on opposition-based grey wolf optimizer algorithm

Anup Kumar Behera[1], Priyanka Agarwal[2]

[1]Department of Mathematics, SRM Institute of Science and Technology, Delhi-NCR Campus, Modinagar, Ghaziabad − 201204 Uttar Pradesh, India.

ab5656@srmist.edu.in, priyankv@srmist.edu.in

Abstract

Software quality assessment depends on software reliability, which is a vital factor. It is tested multiple times before launching software to ensure it is error-free. Software reliability growth models (SRGMs) have proven invaluable for developers and testers providing a framework to scrutinize random failures and ensure software quality. To address this need, there is a crucial need to integrate updated testing effort functions and various fault detection rates into SRGMs, aiming for software that surpasses previous standards. This study introduces a new SRGM, consisting of a Power-law testing effort function (PL-TEF) and a non-linear fault detection rate (FDR), both designed to be more practical in real-world contexts. The model proposed in this research is evaluated using optimization techniques, specifically employing an opposition-based grey wolf optimizer (OBL-GWO) algorithm. This optimization, conducted on a dataset of real failures, validates the efficacy of the model. The results show that the proposed model is the best fit with the data, and the determination of the optimal release time for the proposed model is also calculated. In essence, the positive results derived from this research offer a progressive step forward in the pursuit of superior software quality.

Keywords—NHPP, Software Reliability, Power-law TEF, Non-linear FDR, OBL-GWO.

1. INTRODUCTION

Software developers play a pivotal role by crafting software with robust designs, prioritizing safety, and facilitating easy maintenance to enhance overall software reliability. The reliability of software is influenced by multiple factors, including its design, coding, testing procedures, and ongoing maintenance efforts even after the release of software. It underscores the necessity of careful testing and error resolution before the software is deployed and continuous maintenance and support further contribute to sustaining the reliability and security of the software. Particularly, this factor is critical in industries like aerospace and medical devices, and other domains where stringent safety standards apply. Ultimately, the reliability of software holds paramount importance in meeting user expectations and ensuring the desired outcomes are consistently delivered.

Over the past four decades, many SRGMs have emerged, serving a crucial role in the realm of software testing. These growth models are mathematical frameworks employed for forecasting and evaluating software reliability

as it evolves. These models prove to be highly advantageous in the detection and resolution of defects that may occur during the software development phase.

In 1981, Bev Littlewood et al. proposed an updated version of the first growth model of software reliability, originally developed by Jelinski and Moranda. The authors utilized maximum likelihood estimation (MLE) for parameter estimation to address the instability issue of the J&M model. P.K. Kapur et al. in 1992 utilized the NHP distribution mechanism using a software reliability growth model. The authors assumed that utilizing this technique may uncover previously unnoticed problems without resulting in any program failures. Yamada et al. introduced a growth model that takes into the account test effort required during the testing period. In 1996, Hou et al. utilized the Hyper-Geometric Distribution Software Reliability Model (HGDM) to address two significant challenges in the testing phase. The first challenge was to reduce the number of undetected software faults after testing, while the second challenge was to minimize the required testing resources. They discovered that employing an optimal resource allocation method can enhance the reliability of the software system being examined. In 2001, Kuo et al. attempted to develop an additional growth model utilizing non-homogeneous poisson processes (NHPP). In 2001, Pasquini et al. proposed a novel hypothesis on the failure of software systems. They argued that faults are not solely responsible for component failures but rather that the system as a whole, including human errors, can also contribute to these failures.

In 2007, Huang et al. presented research that involved comparing several software reliability growth models (SRGMs) based on their sensitivity to the effort required to fix faults. The issue at hand was that certain S-shaped models may not perform optimally when faced with varying test efforts. Consequently, a new function was designed to address this concern. The number of reliability growth models was steadily expanding. In 2010, a distance-based model was introduced to aid in identifying the most suitable growth model for reliability testing. This model utilizes input parameters as selection criteria to generate a suitable model. In 2011, Huang et al. proposed a model that distinguishes the speed and number of faults between the debugging processes during the testing and operation phases of software life. They introduced the concept of "multi-points," which are specific points on a time graph that represent changes in the environment. Jain et al. (2020) explored a new method to predict the reliability of the software using the FRF and Gompertz methods to measure the testing effort. According to Kassaymeh et al. (2021), the conventional Backpropagation Neural Network (BPNN) has a limitation where its performance in making accurate estimation is influenced by the initial parameter values, impacting prediction. John et al. (2023) analyzing the reliability of multi-systems, consider different types of failure interactions. It assumes an exponential distribution for the failure and repair and use differential.

Due to the software's resource-intensive nature, the testing process requires a significant allocation of resources, including personnel and resources. Previous studies have demonstrated the suitability of the Weibull, Reileigh, exponential, and other distributions for predicting resource consumption in the modeling of software reliability. This paper applies the NHPP to compute the number of faults and then uses them to estimate the parameters of the software reliability models. Section 2 of this study explains the Power-law testing effort function (PL-TEF).

Section 3 proposes a new framework to obtain the SRGM consideration of power-law TEF and non-linear FDR. Section 4 of the paper provides an in-depth explanation of the OBL-GWO optimization technique. In Section 5, the paper delves into the parameter estimation process within the OBL-GWO algorithm. To conclude, Section 6 offers a summary of the key points discussed in the paper.

1.1 The OBL-GWO algorithm is motivated by the utilization of software reliability growth models:

The OBL-GWO algorithm represents an optimization approach that refines traditional Grey Wolf Optimization (GWO) algorithms through the incorporation of an opposition-based learning (OBL) technique. Its core motivation lies in establishing a more proficient and potent method for parameter estimation within software reliability growth models (SRGMs). In its initial phase, OBL is deployed to generate an initial set of solutions for SRGM parameter estimation, enhancing the speed and robustness of the convergence process in optimization algorithms. Following this, GWO is employed to further refine these initial solutions and identify the optimal solution.

The central goal of the OBL-GWO algorithm revolves around the enhancement of software reliability. This is achieved by amalgamating optimization strategies inspired by the social hierarchy and hunting techniques observed in grey wolves into SRGMs. The emphasis lies on the more effective identification and rectification of software bugs, ultimately contributing to the development of software characterized by heightened reliability.

2. TESTING-EFFORT FUNCTION

A mathematical function that shows how the testing resources are used and changed over time during the software testing phase is called a testing-effort function (TEF). The testing-effort function (TEF) is an essential measure for estimating how much effort is spent in CPU time, labour, human resources, and the number of test cases the software need. It influences the software reliability and the rate of finding faults.

The total amount of testing effort expended in the testing time interval (0,t] is denoted by W(t).

$$W(t) = \int_0^t w(x)\,dx \quad \ldots (1)$$

In this paper, the **power-law (PL)** curve is used to estimate the reliability and its cumulative testing effort consumed in the time (0, t] is

$$W(t) = \frac{\alpha t^{k+1}}{k+1} \quad \ldots (2)$$

and its current testing effort function is

$$w(t) = \alpha t^{k} \quad \ldots (3)$$

Where α is the total expenditure and k is the positive shape parameter.

3. SRGM with PL-TEF and Non-linear FDR

This section discusses the SRG modeling in the context of the power-law testing effort function and non-linear FDR (b(t)). It is also considered that the testing team learns from their experience and becomes more adaptable. Our proposed model is based on the given hypothesis:

1. Failure of the software system occurs randomly due to some fault persisting, and the fault elimination procedure follows the NHPP.
2. When a failure occurs, it is removed promptly, and when no new error is discovered during testing, i.e., debugging is perfect.
3. FDR is considered non-linearly changing and improves efficiency with time. Testers learn from their experience.

Based on the assumptions, we have the mean value function m(t) as

$$\frac{d\,m(t)}{dt}\times\frac{1}{w(t)}=b(t)[a-m(t)] \quad \ldots (4)$$

Where a(t) is the initial fault content function at time t and b(t) is the fault detection rate.

$$b(t)=\begin{cases} b, \text{ FDR is constant} \\ b\beta t^{d}, \text{FDR is non-linear} \end{cases} \quad \ldots (5)$$

On putting the values of b(t) in equation (4), we get the expected removed faults by time t

$$m(t)=\begin{cases} a\left[1-e^{-\alpha b\frac{t^{k+1}}{k+1}}\right], & FDR\ const. \\ a\left[1-e^{-\alpha b\beta\frac{t^{k+d+1}}{k+d+1}}\right], & FDR\ \ non-linear \end{cases} \quad \ldots (6)$$

Serial no.	Model	m(t)
1	Exponential TEF (ETEF)	$m(t)=a[1-e^{-b\alpha(1-e^{-\beta t})}]$
2	Reyleigh TEF (RTEF)	$m(t)=a[1-e^{-b\alpha(1-e^{-\beta t^2})}]$
3	Li and Yi (2016), b(t) is constant (LYM)	Equation (6)
4	Proposed model when b(t) is non linear. (PM)	Equation (6)

Table 1: Models of software reliability and mean value function(m(t))

4. OBL-GWO ALGORITHM

4.1 Grey wolf optimizer algorithm (GWO):

Mirjalili et al. created the GWO algorithm in 2014, drawing inspiration from the social hierarchy and hunting habits of grey wolves. The algorithm sorts the wolves into four ranks reflecting their roles: alpha (α), beta (β), delta (δ), and omega (ω). The alpha wolf is the leader of the pack and represents the best solution. Each grey wolf is under its control. While the beta wolf assists the alpha in decision-making and represents the second-best solution for the whole wolf. The delta wolf, also known as the subordinate wolf, represents the third-best solution for the grey wolves. The remaining wolves belong to the omega wolves. Omega wolves have the lowest score among the wolves. During the hunting, alpha, beta, and delta wolves play a vital role. These three types of wolves

are responsible for tracking, chasing, pursuing, surrounding, and attacking their prey. The hunting mechanism of the grey wolf is divided into three primary steps, which are given below:

- They track, chase, and get closer to the prey.
- They pursue, surround, and harass the prey until it stops moving.
- They attack the target prey.

4.2 Mathematical model of algorithm

The alpha, beta, and delta wolves are the leaders of the hunt. They are essential for tracking, chasing, encircling, and attacking prey. The success of the search is dependent on these three wolves. The mathematical process can be calculated by following equations:

$$\vec{W}(t+1) = \vec{W}_p(t) - \vec{A} \times \vec{D} \quad \text{... (7)}$$

$$\vec{D} = \left| \vec{C} \times \vec{W}_p(t) - \vec{W}(t) \right| \quad \text{... (8)}$$

$$\vec{A} = 2 \times a \times r_1 - a \quad \text{... (9)}$$

$$\vec{C} = 2 \times r_2 \quad \text{... (10)}$$

Where t represents the t-times iteration, $\vec{W}$ and $\vec{W}_p$ represents the position vector of a grey wolf and its prey, $\vec{A}$ and $\vec{C}$ denotes two coefficient vectors, r_1 and r_2 are uniformly generated two random numbers between 0 and 1, the coefficient a is reduced from 2 to 0 during the iteration.

$$a = 2 - \frac{2 \times t}{T} \quad \text{... (11)}$$

Where t represents the current iteration, and T indicates the total number of iterations.

The algorithm updates the wolf's position based on the assumption that alpha, beta, and delta contain better information about potential locations of wolf prey. This suggests that the rest of the pack will modify its position based on the direction of the three dominant wolves. The mathematical process can be calculated by following equations:

$$\left.\begin{aligned} \vec{D}_\alpha &= \left| \vec{C}_\alpha \times \vec{W}_\alpha(t) - \vec{W}(t) \right| \\ \vec{D}_\beta &= \left| \vec{C}_\beta \times \vec{W}_\beta(t) - \vec{W}(t) \right| \end{aligned}\right\} \quad \text{... (12)}$$

$$\vec{D}_\delta = \left| \vec{C}_\delta \times \vec{W}_\delta(t) - \vec{W}(t) \right|$$

$$\left.\begin{aligned} \vec{W}_1(t) &= \vec{W}_\alpha(t) - \vec{A}_\alpha \times \vec{D}_\alpha \\ \vec{W}_2(t) &= \vec{W}_\beta(t) - \vec{A}_\beta \times \vec{D}_\beta \\ \vec{W}_3(t) &= \vec{W}_\delta(t) - \vec{A}_\delta \times \vec{D}_\delta \end{aligned}\right\} \quad \text{... (13)}$$

$$\vec{W}(t+1) = \frac{\vec{W}_1(t) + \vec{W}_2(t) + \vec{W}_3(t)}{3} \quad \text{... (14)}$$

where $\vec{W}_\alpha, \vec{W}_\beta, \vec{W}_\delta$ are positions of the alpha wolf (α), beta wolf (β), delta wolf (δ) in the search space, $\vec{C}_\alpha, \vec{C}_\beta, \vec{C}_\delta$ are three coefficient vectors.

4.3 Opposition-based learning (OBL)

Opposition-based learning (OBL) is an innovative concept in soft computing, introduced by Tizhoosh in 2005, aimed at expediting the convergence rates of meta-heuristic algorithms, particularly in estimating parameters for software reliability growth models. The core incentive behind OBL is to bolster the speed and overall performance of a broad spectrum of optimization techniques in terms of efficiency and effectiveness. The fundamental concept underpinning OBL is the simultaneous consideration of what has been learned by the algorithm over time and the generation of new, random guesses when creating the initial population. As the optimization progresses, introducing an opposite position for each potential solution during the stochastic enhancement phase can significantly improve convergence. This dual-process approach can be initiated at every iteration of the optimization method.

Opposite number

The opposite number ($\ddot{Z}$) for any random values in between *a* and *b* (Z ϵ [*a*, *b*]) can be calculated in the equation (15).

$$\ddot{Z} = a + b - Z \quad \text{... (15)}$$

Where, *a* and *b* are the upper and lower limit of the search space, and Z represents the initial position of the population.

This definition is applicable to higher dimensions as well. Let Z = {z_1, z_2, …,z_n} in a *n*-dimensional search space, where each z_i ($1 \le i \le n$) falls within the range defined by a_i and b_i. To find the corresponding opposite point $\ddot{Z}$= {$\ddot{z}_1$,$\ddot{z}_2$,...$\ddot{z}_n$}, you can use equation (16).

$$\ddot{Z}_i = a_i + b_i - Z_i \quad \ldots(16)$$

4.4 OBL-GWO algorithm

The conventional Grey Wolf Optimisation (GWO) algorithm has a reduced number of parameters for searching and is straightforward to apply. Nevertheless, the experimental results indicate that in certain instances, the exploratory ability of grey wolves is inadequate, leading to a tendency towards local optima. Therefore, there is potential to improve the wolves' ability to explore and make them more effective optimizers. Consequently, we have used a novel equation (Heidari et al. 2019) to investigate the behavior of grey wolves, enabling thorough exploration of the extensive search area. Furthermore, to enhance the efficiency of maintaining convergence speed, an Optimal Baseline Leader (OBL) algorithm, as proposed by Tizhoosh in 2005, is employed to guide the leading wolves in each iteration.

The description of all the applied strategies is as follows:

$$Z^{t+1} = \begin{cases} Z^t_{rand} - r_1 \left| Z^t_{rand} - 2 r_2 Z_{rand} \right|, & r_5 \ge 0.5 \\ \left(Z^t_{alpha} - Z^t_a \right) - r_3 \left(a + r_4 \left(b - a \right) \right), & r_5 \le 0.5 \end{cases} \quad \ldots(17)$$

Where represents the location of the grey wolf in the iteration that follows the current iteration $(t + 1)^{th}$, the vector Z^t_{alpha} represents the position of the alpha wolf at the t^{th} iteration, and the variable Z^t_a represents the average position of grey wolves, In this equation, the current solution is formed based on either a random solution or the best answer.

5. Parameter estimation in OBL-GWO algorithm

The process of parameter estimation involves determining the best-fitting values for unknown parameters in a model based on observed data. In this research, we employ an OBL-GWO algorithm to optimize the parameters of both new and existing models.

This section is to determine the best values that reduce the difference between the real and the predicted faults using the parameters estimated by PSO, GWO, and OBL-GWO. We have utilized a pre-existing dataset of 25 weeks, as mentioned in the work of Chin-Yu Huang et al., and implemented three techniques: PSO, GWO, and OBL-GWO, to estimate a parameter of a software project using four SRGMs: Goel-Okumoto model, Inflection S-shaped model, Delay S-shaped model, and the Proposed model [Table 1]. All models have the same parameter range. We set the population size to 30 and the maximum number of iterations to 500 for PSO, GWO, and OBL-GWO respectively.

Table 2 indicates that OBL-GWO consistently outperforms PSO, GWO, and OBL-GWO in both mean and best solutions across all cases. This underscores the potency of OBL-GWO, demonstrating its ability to circumvent the limitations associated with PSO, GWO, and OBL-GWO, such as local best entrapment and premature convergence. In Figure 1, the optimal convergence graph illustrates the evolutionary process of four models—Exponential TEF, Inflection S-shaped model, Delay S-shaped model, and the proposed model—all employing the OBL-GWO algorithm. Additionally, Figure 2 provides a comparative graph of the actual and estimated failure curves for the four development models. These analyses collectively affirm that the OBL-GWO algorithm consistently delivers superior results compared to conventional PSO, and GWO algorithms, excelling in terms of accuracy, convergence speed, and robustness.

Model	**PSO**			**GWO**			**OBL-GWO**		
	Mean	**SD**	**Best**	**Mean**	**SD**	**Best**	**Mean**	**SD**	**Best**
ETEF	13.9034	1.22	11.62	14.883	8.15	10.09	11.730	0.85	10.66
RTEF	15.5529	1.39	14.26	13.5539	1.61	11.51	11.701	1.16	10.28
LYM	15.4481	2.44	12.04	13.3453	2.04	10.65	12.398	1.96	10.51
PM	12.7285	1.21	11.00	13.0148	2.13	10.06	10.776	0.67	9.23

Table 2.SRGMs with parameters estimated using OBL-GWO-25 weeks measurement.

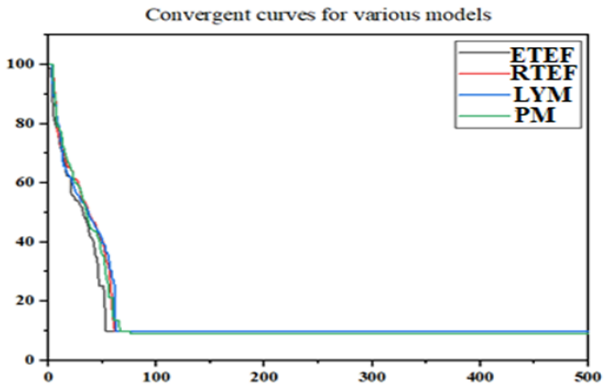

Figure 1: Best convergence for OBL-GWO algorithm applying various models on 25 weeks.

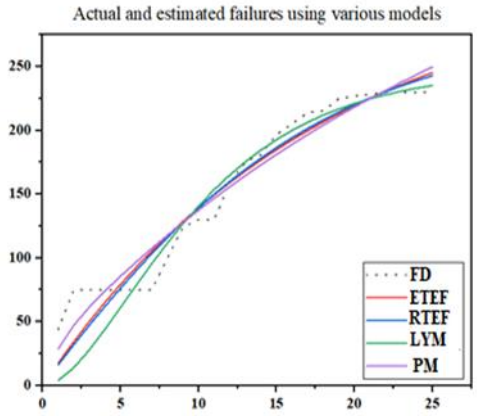

Figure 2: Actual and estimated failure for the OBL-GWO algorithm using various models on 25 weeks.

6. Conclusion

In this study, we examined whether the SRGM was enhanced by integrating a power-law testing effort and a non-linear Fault Detection Rate (FDR) function. The model's performance was evaluated using three optimization algorithms: PSO, GWO, and OBL-GWO, and the result observed that OBL-GWO converges faster than the other two optimization methods. The effectiveness of OBL-GWO has been confirmed in various studies, surpassing PSO and GWO in terms of convergence speed, quality of solutions, and overall robustness. The proposed approach will assist software developers in releasing software at the right time while ensuring the necessary desired reliability. The Non-linear FDR function and PL-TEF may be combined with concepts such as change point phenomena, coverage function, and delay effects in future studies to get more enhance the model's predictive capabilities.

References

1. Aggarwal, Anu G., Abhishek Tandon, and Hoang Pham. Optimization Models in Software Reliability. Springer, 2022.
2. Ahmad, N., and Md Zafar Imam. "Software reliability growth models with Log-logistic testing-effort function: A comparative study." International Journal of Computer Applications 75.12 (2013): 6-11.
3. Dwivedi, Asit, and Deepak Kumar. "Fault dependency SRGM with testing effort using learning function." International Journal of Management Concepts and Philosophy 11.1 (2018): 1-10.
4. Huang, Chin-Yu, and Sy-Yen Kuo. "Analysis of incorporating logistic testing-effort function into software reliability modeling." IEEE Transactions on reliability 51.3 (2002): 261-270.
5. Huang, Chin-Yu, et al. "Software reliability growth models incorporating fault dependency with various debugging time lags." Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.. IEEE, 2004.
6. Jain, M., P. Agarwal, and R. Solanki. "NHPP-based SRGM using time-dependent fault reduction factors (FRF) and Gompertz TEF." Decision Analytics Applications in Industry (2020): 81-89.

7. Jha, P. C., et al. "Optimal testing resource allocation during module testing considering cost, testing effort and reliability." Computers & Industrial Engineering 57.3 (2009): 1122-1130.
8. Jin, Cong, and Shu-Wei Jin. "Parameter optimization of software reliability growth model with S-shaped testing-effort function using improved swarm intelligent optimization." Applied Soft Computing 40 (2016): 283-291.
9. John, Yakubu Mandafiya, et al. "Reliability Analysis of Multi-Hardware–Software System with Failure Interaction." Journal of Computational and Cognitive Engineering 2.1 (2023): 38-46.
10. Kapur, P. K., et al. Software reliability assessment with OR applications. Vol. 364. London: Springer, 2011.
11. Kassaymeh, Sofian, et al. "Salp swarm optimizer for modeling software reliability prediction problems." Neural Processing Letters 53 (2021): 4451-4487.
12. Li, Fan, and Ze-Long Yi. "A new software reliability growth model: multigeneration faults and a power-law testing-effort function." Mathematical Problems in Engineering 2016 (2016).
13. Li, Fan, and Ze-Long Yi. "A new software reliability growth model: multigeneration faults and a power-law testing-effort function." Mathematical Problems in Engineering 2016 (2016).
14. Okumoto, Kazu, and Amrit L. Goel. "Optimum release time for software systems." COMPSAC 79. Proceedings. Computer Software and The IEEE Computer Society's Third International Applications Conference, 1979.. IEEE, 1979.
15. Pham, Hoang. System software reliability. Springer Science & Business Media, 2007.
16. Rafi, Shaik Mohammad, and Shaheda Akthar. "Software reliability growth model with logistic-exponential testing effort function and analysis of software release policy." Proceedings of international conference on advances in computer science. 2010.
17. Yamada, Shigeru, and Hiroshi Ohtera. "Software reliability growth models for testing-effort control." European Journal of Operational Research 46.3 (1990): 343-349.
18. Yamada, Shigeru, Hiroshi Ohtera, and Hiroyuki Narihisa. "A testing-effort dependent software reliability model and its application." Microelectronics Reliability 27.3 (1987): 507-522.
19. Behera, Anup Kumar, and Priyanka Agarwal. "A SOFTWARE RELIABILITY PREDICTION AND MANAGEMENT INCORPORATING CHANGE POINTS BASED ON TESTING EFFORT." Reliability: Theory & Applications 19.2 (78) (2024): 91-100.
20. Behera, Anup Kumar, and Priyanka Agarwal. "An SRGM using Fault Removal Efficiency and Correction Lag Function." International Journal of Reliability, Quality and Safety Engineering (2024).

ABOUT THE AUTHORS

Anup Kumar Behera currently Works as a research scholar in the Mathematics department at SRM Institute of Science and Technology, located in Delhi-NCR Campus, Modinagar, India.

Priyanka Agarwal currently serves as an Assistant Professor in the Mathematics department at SRM Institute of Science and Technology, located in Delhi-NCR Campus, Modinagar, India. She possesses a wealth of knowledge and expertise gained over 17 years of teaching and conducting research. She is particularly interested in researching applied probabilistic models, as well as software and hardware dependability, and inventory modeling.

She has been the recipient of over fifteen national and international publications, four patents, and one granted patent.