Multi-objective Search to Design Multi-phase Model Fitting Algorithms

Joshua Steakelum[1], Jacob Aubertine[1], Kenan Chen[1], Vidhyashree Nagaraju[2], and Lance Fiondella[1]
[1]Electrical and Computer Engineering, University of Massachusetts Dartmouth, MA, USA
[2]Tandy School of Computer Science, University of Tulsa, OK, USA
Email: {jsteakelum, jaubertine, kchen2, and lfiondella}@umassd.edu, vidhyashree-nagaraju@utulsa.edu

***Abstract*—Recent research applies soft computing techniques to fit software reliability growth models. However, runtime performance and the distribution of the distance from an optimal solution over multiple runs must be explicitly considered to justify the practical utility of these approaches, promote comparison, and support reproducible research. This paper presents a meta-optimization framework to design stable and efficient multi-phase algorithms for fitting software reliability growth models. The approach combines initial parameter estimation techniques from statistical algorithms, the global search properties of soft computing, and the rapid convergence of numerical methods. Designs that exhibit the best balance between runtime performance and accuracy are identified. The approach is illustrated on a nonhomogeneous Poisson process software reliability growth model, including a cross-validation step on data sets not used to identify designs. The results indicate the nonhomogeneous Poisson process model considered is too simple to benefit from soft computing because it incurs additional runtime with no increase in accuracy attained.**

***Index Terms*—Software reliability, software reliability growth model, soft computing, numerical methods, multi-phase algorithms**

I. Introduction

Recent research has seen an explosion in the number of studies applying soft computing techniques and especially swarm algorithms [1], [2] to fit software reliability growth models (SRGM). While optimization techniques [3] are essential to fit models and enable predictions, these past studies often fail to consider two competing attributes, namely (i) the speed of convergence to the maximum likelihood estimate (MLE) and (ii) the stability of convergence to this maximum. These two attributes are especially important when implementing tools for non-experts [4], [5] because the model fitting step must be both fast and consistent, so that users who often lack detailed knowledge of the underlying mathematics can be confident that the parameter estimates are accurate and that model assessments and predictions reported by the tool can be trusted. Moreover, soft computing techniques often exhibit robust global search, which has helped to overcome the instability of early numerical techniques such as the Newton-Raphson method. However, numerical methods exhibit mathematically proven rates of convergence and can therefore serve as a powerful complement to soft computing techniques, suggesting that multi-phase algorithms incorporating soft computing techniques followed by traditional optimization procedures may achieve the desired tradeoff between speed and stability of convergence. A framework is needed to identify stable and efficient multi-phase algorithms that leverage the strengths of these alternative approaches to (i) promote the objective comparison of alternative algorithms for fitting models and (ii) focus the research community on the practical goal of stable and efficient algorithmic designs for implementation in a tool that will support the widespread application of SRGM in the user community.

Surveys [6], [7] document dozens of applications of soft computing techniques to fit SRGM and closely related problems, while Mohanty et al. [8] reviewed papers published between 1990 and 2008 that applied AI and soft computing techniques to SRGM, effort estimation, and other software. Examples of machine learning techniques include neural networks [9], [10] and support vector machines [11], [12], while metaphor-based meta-heuristics and evolutionary algorithms include genetic algorithms [13], [14], [15], genetic programming [16], [17], harmony search [18], [19], and gravitational search [20]. Applications of swarm intelligence algorithms, which share information among members of the population, include particle swarm optimization [21], [22], artificial bee colony [23], ant colony optimization [24], [25], cuckoo search [26], grey wolf optimization [27], firefly [28], [29], ant lion optimization [30], and whale optimization [31].

Hybrid approaches have also been proposed, including genetic algorithms to optimize the parameters of particle swarm [32], [33] and grey wolf optimization [34] as well as methods that combine artificial bee colony and particle swarm optimization [35]. Despite their global search properties, neither individual or pairwise combinations of these population-based search techniques converge rapidly and precisely to the maximum in a manner similar to traditional numerical methods such as the Newton-Raphson method when provided accurate initial estimates or statistical algorithms, including the expectation maximization (EM) algorithm [36], [37] and expectation conditional maximization (ECM) algorithm [38], [39]. Thus, multi-phase algorithms composed of a swarm algorithm for global search followed by local search with a numerical or statistical algorithm is a naturally appealing concept to capitalize on the strengths of these two classes of algorithms to achieve a balance between convergence and speed.

To impose structure and support reproducibility, this paper proposes a framework to design multi-phase model fitting algorithms. Algorithms to perform multi-objective optimization [40] such as a posteriori methods, which seek to produce all Pareto optimal solutions or a representative subset, are suitable for this purpose. Examples include Normal Constraint [41] and Successive Pareto Optimization [42] as well as evolutionary methods such as the Non-dominated Sorting Genetic Algorithm (NSGA)-II [43] and the Strength Pareto Evolutionary Algorithm [44]. NSGA-II is employed in this study because of its widespread success in diverse problem domains, although the other approaches may also be suitable for the design of stable, efficient, and accurate multi-phase algorithms.

Our approach explores the space of alternative algorithmic designs for those that exhibit a combination of consistent convergence and runtime. These designs combine initial parameter estimation techniques, swarm algorithms, and numerical methods. The intuition is that designs including a swarm algorithm must contribute to global search without compromising runtime excessively in order to justify the number of iterations, population, and cost of evaluating the objective function and moves within the search space before switching to a gradient-based method. The framework is applied to a nonhomogeneous Poisson process (NHPP) [45] software reliability growth model. A cross-validation step assesses the accuracy and runtime of dominant designs on alternative data sets. The results indicate that the NHPP model did not benefit significantly from a multi-phase algorithm because of the relatively low dimension, smoothness of the objective function, and efficient and accurate initial estimates enabled by the EM algorithm.

The remainder of this paper is organized as follows: Section II describes a concrete implementation of an approach to design multi-phase algorithms. Section III reviews likelihood functions, which serve as the primary optimization objective and Section IV illustrates the approach, including cross-validation to verify speed and stability. Section V concludes and identifies possible directions for future research.

II. Stable, efficient, and accurate multi-phase algorithm design

A systematic approach to design and assess alternative three-phase algorithms would enumerate all possible combinations of algorithms and measure their speed and stability for a range of t_I and t_{II}. Consider two three-phase algorithms A_1 and A_2. These algorithms may be composed of different Phase I, II, and III algorithms as well as potentially distinct rules for determining t_I and t_{II}. These design choices produce unique combinations of stability (percentage of runs that converge to the optimal or a near optimal solution), performance (run time), and accuracy along the Pareto front, where accuracy is defined as $(1 + \varepsilon)$, 1.0 is the optimal value, which is known for test cases, and ε is the error.

A preferred algorithm will be fast, accurate, and stable. However, later t_I may improve stability and accuracy because Phase I algorithms are suitable for global search that increases the likelihood of convergence to the optimal or a near optimal solution, but lower performance because of the computational cost incurred. Conversely, earlier t_I may lower stability and accuracy but improve performance. Similarly, later t_{II} may improve stability an accuracy because Phase II algorithms are suitable for making consistent progress toward a maximum, but also lower performance because of computational costs. Moreover, earlier t_{II} may lower stability and accuracy, but increase performance. Thus, there are many possible algorithmic designs and efficiency is inherently a competing constraint with stability and accuracy.

If algorithms A_1 and A_2 exhibit the same performance but A_1 possesses greater stability, it would be preferred. Similarly, given the choice between two algorithms of equal stability, the faster one would be preferred. Furthermore, a designer may wish to impose an upper bound on the time required to complete and a lower bound on stability to ensure suitability for use in a computer-aided tool. These bounds create constrained multi-objective optimization problems. Algorithms that reside within this region for a range of t_I or t_{II} constitute the space of feasible solutions.

A. *Non-dominated Sorting Genetic Algorithm-II*

NSGA-II [43] is an extension of the genetic algorithm [46] to efficiently identify the Pareto frontier of a multi-objective optimization problem. Inputs include a user specified number of generations (iterations), population of chromosomes (candidate solutions), and a probability of crossover, which is used when hybridizing parent chromosomes to produce candidate offspring. In each generation, chromosomes are decoded and evaluated with respect to a fitness function. In this case, each chromosome represents an alternative multi-phase algorithm design and is run on the optimization problem. One candidate is said to dominate another if and only if all of its optimization objectives are preferred. Each iteration of the NSGA-II algorithm employs non-dominated sorting to order existing candidates according to the number of alternative candidates dominating them.

To avoid outliers, each design is run an odd number of times, non-dominated sorting performed, and the median value chosen as the chromosome's fitness. A crowding distance function is applied to sort chromosomes according to their fitness in a manner that encourages search along the Pareto frontier. Selection samples two pairs of chromosomes and the dominant chromosome in each pair undergo crossover and mutation to produce a pair of offspring. The most dominant parents and offspring combine to form the next generation and the process repeats.

Figure 1 shows the components of an example candidate solution for the multi-phase algorithm design problem. The first three sets of bits respectively correspond to (i) the method of generating initial parameter estimates, (ii) a swarm algorithm to perform efficient global search, and (iii) a numerical method to achieve convergence. Examples of techniques employed to produce initial estimates include interval-constrained random number generation and an adaptation of the expectation maximization algorithm [37]. Swarm algorithms presented in [1],

[2] were implemented, including particle swarm optimization [47], the bat [48], artificial fish swarm [49], cuckoo search [50], firefly [51], flower pollination [52], artificial bee colony [53], and wolf search [54] algorithms. A potential design may also omit the swarm algorithm stage. Swarm bit sequences are used to map uniformly within the range of available swarm algorithms. Our implementation does not allow crossover or mutation within the swarm-stage bits because parameter values that perform well for one algorithm tend not to perform well for another swarm algorithm. This restriction was determined to be reasonable, as phylogenetically diverse animals in nature cannot interbreed. Thus, the population consists of subpopulations that compete for dominance similar to the manner in which different species evolved on earth.

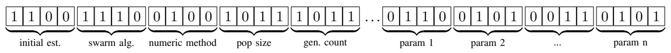

Fig. 1: Encoding of multi-phase algorithm

Numerical methods available in the SciPy library [55], including the Nelder-Mead algorithm [56], Powell's method [57], conjugate gradient [58], the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [59], Limited-memory BFGS [60], truncated Newton's method [61], and the sequential least squares procedure [62] were employed. Numerical bit sequences also map uniformly within the range of available numerical algorithms. However, a numerical method is always included to ensure convergence to an optimum. Thus, the bit sequence $00\ldots0$ encodes a numerical method not the omission of this phase. In all cases, numerical method convergence is defined as $|f(\mathbf{x_{i+1}})-f(\mathbf{x_i})| < \varepsilon$ when improvement in successive values of the objective function fall below a small positive convergence criteria $\varepsilon > 0$.

The fourth and fifth sets of bits are integers defining the population size and number of iterations for the swarm algorithm, if one is included in the design. The remaining sets of bits are floating point values for each parameter of the swarm algorithm. Since the alternative swarm algorithms possess different numbers of parameters, only the first $k < n$ sets of bits are decoded according to the swarm algorithm. This leads to a small amount of memory in unused bits of chromosomes when a swarm algorithm possesses fewer parameters.

To represent the parameters of swarm algorithms as floating point values within a finite interval (θ^-, θ^+), the bit sequence is decoded according to

$$\theta = \theta^- + (\theta^+ - \theta^-) \times \frac{b_{10}}{b_{10}^{\max}} \tag{1}$$

where b_{10} is the Base-10 value of the bit sequence and $b_{10}^{\max}$ is the maximum possible value of that bit sequence. For example, a parameter constrained to the interval $(0.4, 0.5)$ using a four-bit sequence possessing the value 0110_2 may be interpreted as $\theta = 0.44$, since $0.4 + (0.5 - 0.4) \times \frac{6}{15}$. A linear increase in the number of bits exponentially increases the precision of the decimal values in the interval, but also increases the time to decode.

III. Likelihood Function

The multi-phase algorithm design framework is illustrated in the context of nonhomogeneous Poisson process software reliability growth models [45]. Therefore, this section provides a self-contained review. The NHPP counts the number of unique software defects discovered as a function of testing time and a SRGM fit to defect data enables predictions such as the number of defects remaining, the number of defects that would be detected with additional testing, and the probability of failure free operation for a specified period of time in a specified environment (reliability [63]).

Given defect discovery times $\mathbf{T} = \langle t_1, t_2, \ldots, t_n \rangle$, the objective function is to maximize the log-likelihood function

$$LL(t_i; \Theta) = -m(t_n) + \sum_{i=1}^{n} \log\left(\lambda(t_i)\right) \tag{2}$$

where Θ is the vector of model parameters and $\lambda(t) := \frac{\partial m(t)}{\partial t}$ is the instantaneous failure rate at time t.

For example, the mean value function of the Weibull SRGM [64] is

$$m(t) = a\left(1 - e^{-bt^c}\right) \tag{3}$$

where b and c are the scale and shape parameters, respectively.

Two alternative methods for generating initial parameter estimates were incorporated into the encoding of the multi-phase algorithm described in Figure 1. The first was uniform random numbers $b_0 \sim U(0, 0.1)$ and $c_0 \sim U(0, 5)$ with $a_0 = \frac{n}{1-e^{-b_0 t^{c_0}}}$ and the second was uniform random numbers in an interval about feasible initial estimates determined by the expectation maximization algorithm [37], such that $b_0 \sim U(\frac{1}{s} \times \frac{n}{\sum_{i-1}^{n} t_i^c}, s \times \frac{n}{\sum_{i-1}^{n} t_i^c})$ and $c_0 \sim U(\frac{1}{s}, s)$, since $c = 1$ is the special case where the Weibull reduces to the Goel-Okumoto [65] model and $s = 2$ is a user-defined parameter to control the width of the interval.

IV. Illustrations

This section illustrates the application of the framework to the nonhomogeneous Poisson process, including design and cross-validation experiments and accompanying discussion.

Table I summarizes the parameters of NSGA-II, including the number of generations, population size, precision of numerical parameters, and crossover logic.

TABLE I: NSGA-II Parameters

Parameter	Value
Generations	128
Population	128
Number of runs	31
Bits per parameter	32
Crossover probability	98%
Use head-tail crossover	True

A. NHPP Software Reliability Growth Model

The NSGA-II implementation of the multi-phase algorithm design problem was run with the Weibull NHPP SRGM as the objective function defined by Equations (2) and (3) on the SYS1 data set [66]. Pareto optimal designs trained on this data set were then cross-validated with eight similar data sets to assess the generalizability of their performance with respect to run time and accuracy.

Figure 2 shows the percentage of the population utilizing each swarm algorithm as a function of the generations of NSGA-II. By the end of 128 iterations, designs incorporating the artificial bee colony (ABC) algorithm constituted the majority of the population. However, this does not necessarily mean that multi-phase designs incorporating ABC are "best" because the multi-objective nature of the problem requires explicit consideration of the tradeoff between speed and accuracy, which we were able to quantify because the maximum of Equation (2) on the SYS1 data [66] is known to be 966.0803 at parameter values $\hat{a} = 172.5262$, $\hat{b} = 0.000696$, and $\hat{c} = 0.676739$. This design stage required about 90 minutes to complete.

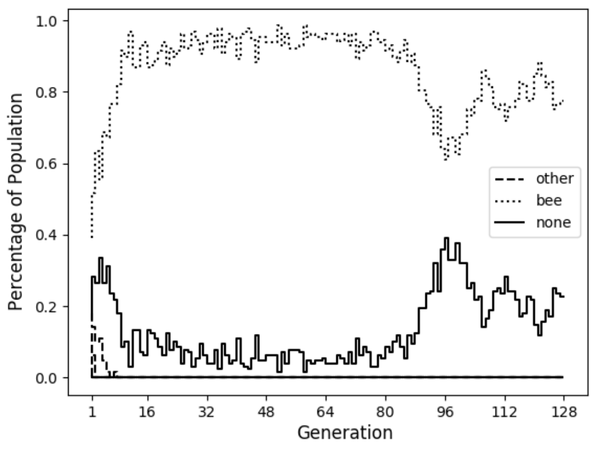

Fig. 2: Percentage of population utilizing alternative swarm algorithms in each generation of NSGA-II

To compare alternative designs, Figure 3 shows a Pareto plot composed of the 128 members of the population in the final generation. Figure 3 indicates that the designs achieving the lowest median runtimes included a swarm stage utilizing the artificial bee colony algorithm, but a median error of as much as 0.8%, whereas many of the designs achieving low median error did not incorporate a swarm stage. For example, the fastest design (0.001990 seconds) with low median error ($\varepsilon = 0.005716$), denoted Design 1, employed six iterations of the artificial bee colony algorithm with a population of six bees with subpopulations scout (34.7%), experienced (39.5%), and onlooker (25.8%), which rounded to two bees in each of the three subpopulations, and experienced bee parameters $w_b = 0.607$, $w_g = 0.738$, $r = 0.191$. This brief swarm stage was followed by the Truncated Newton algorithm.

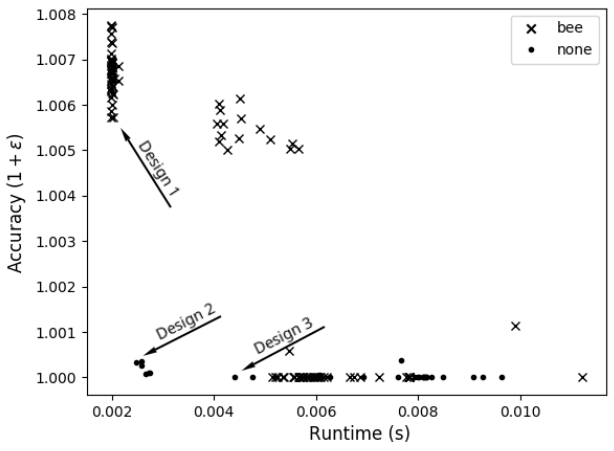

Fig. 3: Pareto frontier of multi-phase algorithm designs in final generation of NSGA-II on Weibull NHPP SRGM

Solutions at the knee of the curve in the bottom left of Figure 3 may be preferred because they simultaneously achieve low error and runtime. For example, Design 2 was the fastest algorithm with error below 0.001 ($\varepsilon = 0.000273$), achieving a median runtime of 0.002581 seconds with the Broyden-Fletcher-Goldfarb-Shanno algorithm and no swarm stage. Design 3 also employed BFGS, exhibiting median error nearly 16 times smaller than Design 2 with $\varepsilon = 0.000017$, but increased median runtime approximately 1.67 times to 0.004414 seconds. Thus, the variation between Designs 2 and 3 was explained by the number of runs (31) and initial parameter estimation based on the EM algorithm, which was selected over uniform random numbers in all three designs identified in Figure 3.

To test the generalizability of our designs, we ran the two unique designs identified in Figure 3 on eight additional failure times data sets from the software reliability literature [66], which took about five minutes to complete. The number of runs was increased to 63 and the 32^{nd} run from the dominated sort used to determine the median, which greatly reduced variation similar to the differences between Designs 2 and 3 observed in the design phase.

Figure 4 shows the results of these cross-validation experiments, where each of the eight data sets is indicated by a unique marker and the results of Design 1 (black), which applied EM initial estimates, six iterations of the artificial bee colony optimization, and a Truncated Newton algorithm, while Design 2/3 (gray) applied EM initial estimates and the Broyden-Fletcher-Goldfarb-Shanno algorithm. With few exceptions, nearly 90% (24/27) of the combinations exhibited median error below 0.001. Moreover, most of the combinations with negligible error exhibited median run times between 0.005 and 0.015 seconds, similar to the range $(0.005, 0.010)$ observed in Figure 3. The relatively slow performance of both designs on CSR1 is likely data set specific. The main result of the comparison is that Design 1 possessing a swarm stage was only faster on one data set (S2) and more accurate on only two data sets (SYS2 and SS3), suggesting that a simple and efficient initial parameter estimation technique applied in conjunction with a traditional numerical method may be preferred over a multi-phase algorithm utilizing a swarm stage for models with a relatively small number of parameters.

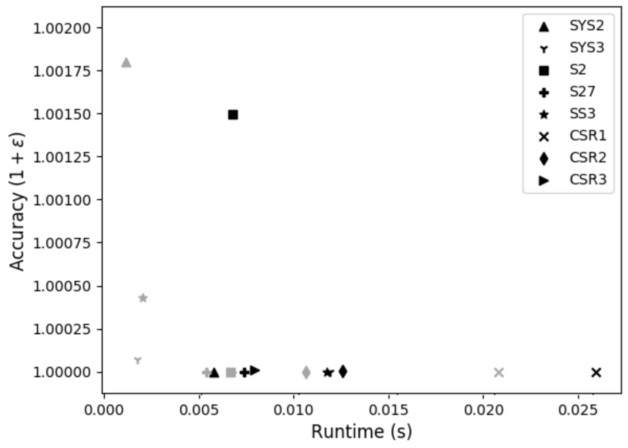

Fig. 4: Cross-validation Pareto frontier

V. Conclusions and future research

This paper presented a framework to design stable and efficient multi-phase algorithms for fitting software reliability growth models. The Non-dominated Sorting Genetic Algorithm-II was employed to identify designs that achieved a desirable tradeoff between these competing objectives. The framework implemented several swarm intelligence algorithms and numerical methods, allowing designs with and without a swarm stage. The framework was employed to identify multi-phase algorithms for a nonhomogeneous Poisson process software reliability growth model. Cross-validation was performed on other failure time data sets. The results suggested that the NHPP SRGM considered did not benefit significantly from a multi-phase algorithm.

To promote future research, the source code of the framework, algorithms implemented, and experiments have been published as an open source repository, available from GitHub. The framework will be applied to higher dimensional problems such as models possessing a larger number of parameters.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant Number (1749635). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] A. Hassanien and E. Emary, *Swarm Intelligence: Principles, Advances, and Applications*. CRC Press, 2016.

[2] X.-S. Yang, *Nature-inspired Metaheuristic Algorithms*. Elsevier, 2014.

[3] F. Archetti and F. Schoen, "A survey on the global optimization problem: general theory and computational approaches," *Annals of Operations Research*, vol. 1, no. 2, pp. 87–110, 1984.

[4] M. Lyu and A. Nikora, "CASRE - a computer-aided software reliability estimation tool," in *Proc. of Computer-Aided Software Engineering Workshop*, Montreal, Canada, jul 1992, pp. 264–275.

[5] K. Shibata, K. Rinsaka, and T. Dohi, "M-srat: Metrics-based software reliability assessment tool," *International Journal of Performability Engineering*, vol. 11, no. 4, pp. 369–379, 2015.

[6] K. Kaswan, S. Choudhary, and K. Sharma, "Software reliability modeling using soft computing techniques: Critical review," *Journal of Information Technology and Software Engineering*, vol. 5, p. 144, 2015.

[7] A. Hudaib and M. Moshref, "Survey in software reliability growth models: Parameter estimation and models ranking," *International Journal of Computer Systems*, vol. 5, no. 5, pp. 11–25, 2018.

[8] R. Mohanty, V. Ravi, and M. R. Patra, "The application of intelligent and soft-computing techniques to software engineering problems: A review," *International Journal of Information and Decision Sciences*, vol. 2, no. 3, pp. 233–272, 2010.

[9] N. Karunanithi, D. Whitley, and Y. K. Malaiya, "Prediction of software reliability using connectionist models," *IEEE Transactions on software engineering*, vol. 18, no. 7, pp. 563–574, 1992.

[10] T. Dohi, Y. Nishio, and S. Osaki, "Optimal software release scheduling based on artificial neural networks," *Annals of Software Engineering*, vol. 8, 1999.

[11] F. Xing and P. Guo, "Support vector regression for software reliability growth modeling and prediction," in *Proc. Advances in Neural Networks*, 2005, pp. 925–930.

[12] P.-F. Pai and W.-C. Hong, "Software reliability forecasting by support vector machines with simulated annealing algorithms," *Journal of Systems and Software*, vol. 79, pp. 747–755, 2006.

[13] T. Minohara and Y. Tohma, "Parameter Estimation of Hyper-Geometric Distribution Software Reliability Growth Model by Genetic Algorithms," in *Proc. International Symposium on Software Reliability Engineering*, Toulouse, France, oct 1995, pp. 324–329.

[14] M. Chen, H. Wu, and H. Shyur, "Analyzing software reliability growth model with imperfect-debugging and changepoint by genetic algorithms," in *Proc. International Conference on Computers and Industrial Engineering*, Montreal, Canada, nov 2001, pp. 520–526.

[15] Y. Dai, M. Xie, K. Poh, and B. Yang, ""Optimal testing resource allocation with genetic algorithm for modular software systems"," *Journal of Systems and Software*, vol. 66, no. 1, pp. 47–55, 2003.

[16] E. O. Costa, S. R. Vergilio, A. T. R. Pozo, and G. A. Souza, "Modeling software reliability growth with genetic programming," in *Proc. International Symposium on Software Reliability Engineering*, Chicago, IL, nov 2005.

[17] E. O. Costa, G. A. de Souza, A. T. R. Pozo, and S. R. Vergilio, "Exploring genetic programming and boosting techniques to model software reliability," *IEEE Transactions on Reliability*, vol. 56, no. 3, pp. 422–434, 2007.

[18] I. Altaf, I. Majeed, and K. Arshid Iqbal, "Effective and optimized software reliability prediction using harmony search algorithm," in *Proc. International Conference on Green Engineering and Technologies*, 2016, pp. 1–6.

[19] A. Choudhary, A. S. Baghel, and O. P. Sangwan, "Efficient parameter estimation of software reliability growth models using harmony search," *IET Software*, vol. 11, no. 6, pp. 286–291, 2017.

[20] ——, "An efficient parameter estimation of software reliability growth models using gravitational search algorithm," *International Journal of System Assurance Engineering and Management*, vol. 8, no. 1, pp. 79–88, mar 2017.

[21] A. Sheta, "Reliability growth modeling for software fault detection using particle swarm optimization," in *Proc. IEEE Congress on Evolutionary Computation*, 2006, pp. 3071–3078.

[22] C. Jin and S.-W. Jin, "Parameter optimization of software reliability growth model with s-shaped testing-effort function using improved swarm intelligent optimization," *Applied Soft Computing*, vol. 40, pp. 283–291, 2016.

[23] D. T. Sharma, M. Pant, and A. Abraham, “Dichotomous search in abc and its application in parameter estimation of software reliability growth models,” in *Proc. World Congress on Nature and Biologically Inspired Computing*, 2011, pp. 207–212.

[24] C. Zheng, X. Liu, S. Huang, and Y. Yao, “A parameter estimation method for software reliability models,” *Procedia engineering*, vol. 15, pp. 3477–3481, 2011.

[25] L. Shanmugam and L. Florence, “A comparison of parameter best estimation method for software reliability models,” *International Journal of Software Engineering and Applications*, vol. 3, pp. 91–102, 2012.

[26] D. Al-Saati and M. Abd-AlKareem, “The use of cuckoo search in estimating the parameters of software reliability growth models,” *International Journal of Computer Science and Information Security*, vol. 11, 2013.

[27] A. Sheta and A. Abdel-Raouf, “Estimating the parameters of software reliability growth models using the grey wolf optimization algorithm,” *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 4, pp. 499–505, 2016.

[28] N. Al-Saati and M. Alabajee, “On the performance of firefly algorithm in software reliability modeling,” *Proc. International Journal of Recent Research and Review*, vol. 9, no. 4, pp. 1–9, 2016.

[29] A. Choudhary, A. S. Baghel, and O. P. Sangwan, “Parameter estimation of software reliability model using firefly optimization,” in *Proc. Data Engineering and Intelligent Computing*. Springer, 2018, pp. 407–415.

[30] M. Alabajee and T. Alreffaee, “Exploring ant lion optimization algorithm to enhance the choice of an appropriate software reliability growth model,” *International Journal of Computer Applications*, vol. 182, no. 4, pp. 1–8, 2018.

[31] K. Lu and Z. Ma, “Parameter estimation of software reliability growth models by a modified whale optimization algorithm,” in *Proc. IEEE International Symposium on Distributed Computing and Applications for Business Engineering and Science*, 2018, pp. 268–271.

[32] M. Rao and K. Anuradha, “A hybrid method for parameter estimation of software reliability growth model using modified genetic swarm optimization with the aid of logistic exponential testing effort function,” in *Proc. IEEE International Conference on Research Advances in Integrated Navigation Systems*, 2016, pp. 1–8.

[33] A. Kumar, R. P. Tripathi, P. Saraswat, and P. Gupta, “Parameter estimation of software reliability growth models using hybrid genetic algorithm,” in *Proc. IEEE International Conference on Image Information Processing*, 2017, pp. 1–6.

[34] J. Alneamy and M. Dabdoob, “The use of original and hybrid grey wolf optimizer in estimating the parameters of software reliability growth models,” *International Journal of Computer Applications*, vol. 167, no. 3, pp. 12–21, 2017.

[35] Z. Li, M. Yu, D. Wang, and H. Wei, “Using hybrid algorithm to estimate and predicate based on software reliability model,” *IEEE Access*, vol. 7, pp. 84 268–84 283, 2019.

[36] H. Okamura, Y. Watanabe, and T. Dohi, “Estimating mixed software reliability models based on the EM algorithm,” in *Proc. Proceedings International Symposium on Empirical Software Engineering*, oct 2002, pp. 69–78.

[37] ——, “An iterative scheme for maximum likelihood estimation in software reliability modeling,” in *Proc. International Symposium on Software Reliability Engineering*, nov 2003, pp. 246–256.

[38] P. Zeephongsekul, C. Jayasinghe, L. Fiondella, and V. Nagaraju, “Maximum-likelihood estimation of parameters of NHPP software reliability models using expectation conditional maximization algorithm,” *IEEE Transactions on Reliability*, vol. 65, no. 3, pp. 1571–1583, 2016.

[39] V. Nagaraju, L. Fiondella, P. Zeephongsekul, C. Jayasinghe, and T. Wandji, “Performance optimized expectation conditional maximization algorithms for nonhomogeneous Poisson process software reliability models,” *IEEE Transactions on Reliability*, vol. 66, no. 3, pp. 722–734, 2017.

[40] K. Deb, *Multi-Objective Evolutionary Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 995–1015. [Online]. Available: https://doi.org/10.1007/978-3-662-43505-2_49

[41] A. Messac, A. Ismail-Yahaya, and C. Mattson, “The normalized normal constraint method for generating the pareto frontier,” *Structural and Multidisciplinary Optimization*, vol. 25, no. 2, pp. 86–98, Jul. 2003.

[42] D. Mueller-Gritschneder, H. Graeb, and U. Schlichtmann, “A successive approach to compute the bounded pareto front of practical multiobjective optimization problems,” *SIAM Journal on Optimization*, vol. 20, no. 2, pp. 915–934, Jan. 2009.

[43] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, apr 2002.

[44] E. Zitzler, M. Laumanns, and L. Thiele, “Spea2: Improving the strength pareto evolutionary algorithm,” *TIK-report*, vol. 103, 2001.

[45] W. Farr, “Software reliability modeling survey,” in *M. Lyu (ed): Handbook of Software Reliability Engineering*. McGraw-Hill, 1996, vol. 222, pp. 71–117.

[46] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 2012.

[47] R. Eberhart and J. Kennedy, “A new optimizer using particle swarm theory,” in *Proc. IEEE International Symposium on Micro Machine and Human Science*, Nagoya, Japan, oct 1995, pp. 39–43.

[48] X.-S. Yang, *A New Metaheuristic Bat-Inspired Algorithm*. Springer Berlin Heidelberg, 2010, vol. Nature Inspired Cooperative Strategies for Optimization, pp. 65–74.

[49] X.-L. Li, Z.-J. Shao, and J.-X. Qian, “An optimizing method based on autonomous animats: Fish-swarm algorithm,” *Systems Engineering - Theory & Practice*, vol. 22, no. 11, pp. 32–38, 2002.

[50] X. Yang and Suash Deb, “Cuckoo search via Lévy flights,” in *Proc. World Congress on Nature Biologically Inspired Computing*, dec 2009, pp. 210–214.

[51] X.-S. Yang, “Firefly algorithms for multimodal optimization,” in *International symposium on stochastic algorithms*. Springer, 2009, pp. 169–178.

[52] ——, “Flower pollination algorithm for global optimization,” in *Proc. International Conference on Unconventional Computing and Natural Computation*. Springer, 2012, pp. 240–249.

[53] D. Karaboga and B. Basturk, “A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm,” *Journal of Global Optimization*, vol. 39, no. 3, pp. 459–471, 2007.

[54] R. Tang, S. Fong, X. Yang, and S. Deb, “Wolf search algorithm with ephemeral memory,” in *Proc. International Conference on Digital Information Management*, aug 2012, pp. 165–172.

[55] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and C. ontributors SciPy 1.0, “SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python,” *arXiv e-prints*, p. arXiv:1907.10121, jul 2019.

[56] J. A. Nelder and R. Mead, “A simplex method for function minimization,” *The Computer Journal*, vol. 7, no. 4, pp. 308–313, Jan. 1965.

[57] M. J. D. Powell, “An efficient method for finding the minimum of a function of several variables without calculating derivatives,” *The Computer Journal*, vol. 7, no. 2, pp. 155–162, Feb. 1964.

[58] M. Hestenes and E. Stiefel, “Methods of conjugate gradients for solving linear systems,” *Journal of Research of the National Bureau of Standards*, vol. 49, no. 6, p. 409, Dec. 1952.

[59] R. Fletcher, *Practical methods of optimization*. John Wiley & Sons, 1986.

[60] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, “A limited memory algorithm for bound constrained optimization,” *SIAM Journal on Scientific Computing*, vol. 16, no. 5, pp. 1190–1208, Sep. 1995.

[61] L. Grippo, F. Lampariello, and S. L. I. L, “A truncated newton method with nonmonotone line search for unconstrained optimization,” *Journal of Optimization Theory and Applications*, vol. 60, no. 3, pp. 401–419, 1989.

[62] J.-F. Bonnans, J. C. Gilbert, C. Lemaréchal, and C. A. Sagastizábal, *Numerical optimization theoretical and practical aspects*. Springer, 2006.

[63] “Standard glossary of software engineering terminology (std-729-1991),” ANSI/IEEE, Tech. Rep., 1991.

[64] S. Yamada and S. Osaki, “Reliability growth models for hardware and software systems based on nonhomogeneous Poisson processes: A survey,” *Microelectronics Reliability*, vol. 23, no. 1, pp. 91–112, 1983.

[65] A. Goel, “Software reliability models: Assumptions, limitations, and applicability,” *IEEE Transactions on Software Engineering*, vol. SE-11, no. 12, pp. 1411–1423, Dec. 1985. [Online]. Available: https://doi.org/10.1109/tse.1985.232177

[66] M. Lyu, Ed., *Handbook of Software Reliability Engineering*. New York, NY: McGraw-Hill, 1996.