Annals of Operations Research
https://doi.org/10.1007/s10479-021-04028-w

S.I.: RELIABILITY MODELING WITH APPLICATIONS BASED ON BIG DATA

Multi-phase algorithm design for accurate and efficient model fitting

Joshua Steakelum[1] · Jacob Aubertine[1] · Kenan Chen[1] · Vidhyashree Nagaraju[1] · Lance Fiondella[1]

Accepted: 3 March 2021

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 through nonhomogeneous Poisson process and covariate software reliability growth models, 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. However, a multi-phase design for the covariate software reliability growth model consisting of the bat algorithm followed by a numerical method achieves better performance and converges consistently, compared to a numerical method only. The proposed approach supports higher-dimensional covariate software reliability growth model fitting suitable for implementation in a tool.

Keywords Software reliability · Software reliability growth model · Soft computing · Numerical methods · Multi-phase algorithms

1 Introduction

Recent research has seen an explosion in the number of studies applying soft computing techniques and especially swarm algorithms (Hassanien and Emary 2016; Yang 2014) to fit

✉ Lance Fiondella
lfiondella@umassd.edu

[1] University of Massachusetts Dartmouth, 285 Old Westport Road, North Dartmouth, MA 02747, USA

software reliability growth models (SRGM). While optimization techniques (Archetti and Schoen 1984) 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 (Lyu and Nikora 1992; Shibata et al. 2015) 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 (Hudaib and Moshref 2018; Kaswan et al. 2015) document dozens of applications of soft computing techniques to fit SRGM and closely related problems, while (Mohanty et al. 2010) 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 (Dohi et al. 1999; Karunanithi et al. 1992) and support vector machines (Pai and Hong 2006; Xing and Guo 2005), while metaphor-based meta-heuristics and evolutionary algorithms include genetic algorithms (Chen et al. 2001; Dai et al. 2003; Minohara and Tohma 1995), genetic programming (Costa et al. 2005, 2007), harmony search (Altaf et al. 2016; Choudhary et al. 2017b), and gravitational search (Choudhary et al. 2017a). Applications of swarm intelligence algorithms, which share information among members of the population, include particle swarm optimization (Jin and Jin 2016; Sheta 2006), artificial bee colony (Sharma et al. 2011), ant colony optimization (Shanmugam and Florence 2012; Zheng et al. 2011), cuckoo search (Al-Saati and Abd-AlKareem 2013), grey wolf optimization (Sheta and Abdel-Raouf 2016), firefly (Al-Saati and Alabajee 2016; Choudhary et al. 2018), ant lion optimization (Alabajee and Alreffaee 2018), and whale optimization (Lu and Ma 2018).

Hybrid approaches have also been proposed, including genetic algorithms to optimize the parameters of particle swarm (Kumar et al. 2017; Rao and Anuradha 2016) and grey wolf optimization (Alneamy and Dabdoob 2017) as well as methods that combine artificial bee colony and particle swarm optimization (Li et al. 2019). 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 (Okamura et al. 2002, 2003) and expectation conditional maximization (ECM) algorithm (Nagaraju et al. 2017; Zeephongsekul et al. 2016). 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 (Deb 2015) 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 (Messac et al. 2003) and Successive Pareto Optimization (Mueller-Gritschneder et al. 2009) as well as evolutionary methods such as the Non-dominated Sorting Genetic Algorithm (NSGA)-II (Deb et al. 2002) and the Strength Pareto Evolutionary Algorithm (Zitzler et al. 2001). 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 nonhomogeneous Poisson process (NHPP) (Farr 1996) and covariate (Shibata et al. 2006) software reliability growth models. 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. However, the best multi-phase algorithm incorporating a swarm stage performed demonstrably better than the next best alternative, which only utilized a numerical method, because of the higher dimension and greater degree of nonlinearity in the objective function as well as the lack of a fast and accurate method to estimate the initial parameters. Therefore, the proposed framework can be used to identify algorithmic designs which are suitable for the complexity of the problem and may exhibit enhanced performance on models with a larger number of covariates. Therefore, the present paper addresses a key technical challenge, namely enabling the application of covariate SRGM to higher dimensional data in a manner that is both stable and efficient. These results also possess practical implications, since the designs identified are suitable for implementation in a tool that promotes adoption of covariate models by non-experts who collect software reliability and security metrics, so that they can assess the effectiveness of their processes.

The remainder of this paper is organized as follows: Section 2 describes the motivation and conceptual framework for multi-phase algorithmic design. Section 3 describes a concrete implementation of an approach to design multi-phase algorithms, while Sect. 4 summarizes the swarm intelligence algorithms employed by the framework. Section 5 reviews likelihood functions, which serve as the primary optimization objective and Sect. 6 illustrates the approach, including cross-validation to verify speed and stability. Section 7 concludes and identifies possible directions for future research.

2 Multi-phase algorithms

Figure 1 depicts the concept of a multi-phase algorithm with an example of a three-phase algorithm to identify the maximum likelihood estimates of a model's parameters.

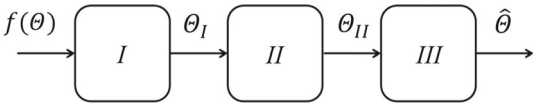

Fig. 1 Three-phase algorithm: modular design and parameters

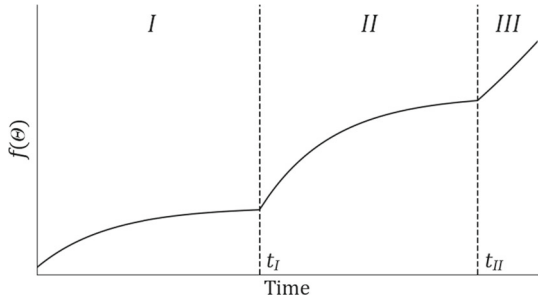

Fig. 2 Three-phase algorithm: conceptual view of convergence

Input $f(\Theta)$ is a parametric function. Output $\hat{\Theta}$ is the vector of numerical parameters that optimize $f(\Theta)$ such as the maximum likelihood estimates of a software reliability model corresponding to the values that best fit a data set. Θ_I and Θ_{II} are the intermediate parameter estimates output from phases I and II.

A desirable characteristic of a Phase I algorithm is robust global search of the multi-dimensional space of parameter values in order to identify good initial parameter estimates Θ_I. Swarm algorithms are appropriate for Phase I because they can quickly identify a near optimal solution. However, Phase I algorithms alone are undesirable because randomness in their search procedures slows convergence as time progresses. Thus, initial parameter estimates Θ_I identified in Phase I should be input into a Phase II algorithm that possesses greater stability, enabling consistent progress toward the optimum. The EM and ECM algorithms are suitable Phase II algorithms because they improve monotonically, but should not be used in isolation because they can converge slowly. Hence, intermediate parameter estimates Θ_{II} can be passed to a Phase III algorithm, which will converge rapidly. Therefore, Newton's method and other classical optimization (Bertsekas 1999) algorithms are suitable for Phase III.

Figure 2 offers a conceptual view of a three-phase algorithm, depicting convergence toward a maximum and times t_I and t_{II} at which the algorithm transitions from one phase to the next. In addition to the choice of algorithm for each phase, an additional consideration is the choice of t_I and t_{II} to achieve a compromise between speed and stability. Divergence may result when switching phases too soon, whereas waiting longer will lower performance. Thus, the choice of t_I and t_{II} impose tradeoffs between speed and stability.

Combinations of two phases may also yield algorithms with competitive speed and stability. Therefore, different combinations of two phases are also considered. Table 1 shows the seven possible combinations of algorithms, where an X indicates inclusion of a phases from Fig. 1. Table 1 also identifies references to combinations previously considered, while combinations with no reference indicate little or no previous research on that combination. Examples of combinations 1, 2, and 3 include Newton's method (Goel 1982), the EM algorithm (Okamura et al. 2003) and particle swarm optimization (Sheta 2006) respectively. Combination 4–6 correspond to (4) EM or ECM algorithms coupled with classical optimization (Nagaraju et al. 2017), (5) swarm algorithms with classical optimization methods, and (6) swarm algorithms coupled with the EM or ECM algorithm, while combination (7) considers all three phases described in Figs. 1 and 2.

Table 1 Combination of phases

Combination #	I	II	III	References
1			*X*	Goel (1982)
2		*X*		Okamura et al. (2003)
3	*X*			Sheta (2006)
4		*X*	*X*	Nagaraju et al. (2017)
5	*X*		*X*	
6	*X*	*X*		
7	*X*	*X*	*X*	

3 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. Ultimately, our experimental framework implemented combination (5) of Table 1 because EM and ECM algorithms proved to be relatively slow and a swarm algorithm was often sufficient to identify initial estimates for a numerical method.

3.1 Non-dominated sorting genetic algorithm-II

NSGA-II (Deb et al. 2002) is an extension of the genetic algorithm (Goldberg 2012) 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

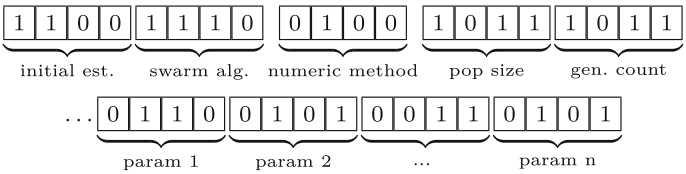

Fig. 3 Encoding of multi-phase algorithm

evaluated with respect to a fitness function. In this case, each chromosome represents an alternative multi-phase algorithm design, including several of the alternatives described in Table 1, 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 3 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 (Okamura et al. 2003). Swarm algorithms presented in (Hassanien and Emary 2016; Yang 2014) were implemented, including particle swarm optimization (Eberhart and Kennedy 1995), the bat (Yang 2010), artificial fish swarm (Li et al. 2002), cuckoo search (Yang 2009), firefly (Yang 2009), flower pollination (Yang 2012), artificial bee colony (Karaboga and Basturk 2007), and wolf search (Tang et al. 2012) algorithms. Furthermore, setting all of the bits in this set to zero omits a swarm algorithm from the multistage design. Moreover, when the number of alternative swarm algorithms or numerical methods is not a power of two, initial bit sequences above the valid range are set uniformly at random to a number within the valid integer range. Our implementation does not allow crossover or mutation with the swarm 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.

Numerical methods available in the SciPy library (Virtanen et al. 2019), including the Nelder-Mead algorithm (Nelder and Mead 1965), Powell's method (Powell 1964), conjugate gradient (Hestenes and Stiefel 1952), the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm (Fletcher 1986), Limited-memory BFGS (Byrd et al. 1995), truncated Newton's method (Grippo et al. 1989), and the sequential least squares procedure (Bonnans et al. 2006) were employed. When the number of available numerical algorithms is not a power of two, initial and mutated bit sequences above the valid range also map uniformly at random to one of the available 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.

4 Swarm intelligence algorithms for multi-phase numerical methods

This section provides a self-contained summary of the swarm algorithms implemented, including particle swarm optimization, the bat algorithm, artificial fish swarm, cuckoo search, the firefly and flower pollination algorithms as well as artificial bee colony optimization.

4.1 Particle swarm optimization (PSO)

In particle swarm optimization (Eberhart and Kennedy 1995), particles possess velocities and locations. The velocity vector of the i^{th} particle at discrete time step $(t + 1)$ is

$$\mathbf{v}_i^{t+1} = \omega \mathbf{v}_i^t + c_1 \boldsymbol{\beta}_1 (\mathbf{p}_i - \mathbf{x}_i^t) + c_2 \boldsymbol{\beta}_2 (\mathbf{g} - \mathbf{x}_i^t) \tag{2}$$

where ω, c_1, and c_2, all > 0, are user specified constants, while $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2 \sim U(0, 1)$ are uniform random vectors generated at each time step to introduce randomness into the search process. Thus, the velocity update equation is a weighted sum of the present velocity $(\mathbf{v}_i^t)$ and vectors pointing from the particle's present position to the direction of the particle best $\left(\mathbf{p}_i - \mathbf{x}_i^t\right)$ and global best $\left(\mathbf{g} - \mathbf{x}_i^t\right)$ respectively.

The position of the ith particle at time step $(t + 1)$ is simply the sum of the particle's present position and velocity in time step $(t + 1)$ or

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1} \tag{3}$$

A large number of extensions to PSO (Bansal et al. 2011) modify the inertia term ω.

4.2 Bat algorithm (BA)

The Bat Algorithm (Yang 2010) abstracts microbat echolocation to track prey with sound pulses. As time passes and bats approach their prey, the loudness (A) of sound pulses decreases and the rate of emissions (r) increases.

The velocity vector of the i^{th} bat at time step $(t+1)$ is

$$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + (\mathbf{x}_i^t - \mathbf{x}_*) f_i^{t+1} \tag{4}$$

where $\mathbf{x}_i^t$ and $\mathbf{x}_*$ are the present position of the i^{th} bat and the global best, while

$$f_i^{t+1} = f_{min} + (f_{max} - f_{min})\beta_1 \tag{5}$$

is the frequency with $f_{\max} > f_{\min} \geq 0$, and $\beta_1 \sim U(0,1)$. Thus, $f_i^{t+1} \sim U(f_{min}, f_{max})$, promoting global search away from the global best.

In each iteration, the new position vector is

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1} \tag{6}$$

It is also possible that a bat performs local search if $\beta_2 \sim U(0,1)$ is greater than r_i^t, where

$$r_i^t = r_i^0(1 - e^{-\gamma(t-1)}) \tag{7}$$

is the present pulse emission rate of the i^{th} bat, with initial pulse emission rate $r_i^0 \in (0,1)$ and $\gamma > 0$. Thus, local search is more likely in earlier iterations, since r increases exponentially.

A bat performing local search explores a random location near the global best according to

$$\mathbf{x}_i^{local} = \mathbf{x}_* + \boldsymbol{\epsilon} A^t \tag{8}$$

where $\boldsymbol{\epsilon} \sim U(-1,1)$ and A^t is the average loudness of all bats. The loudness of the i^{th} bat at time step $(t+1)$ is

$$A_i^{t+1} = \alpha A_i^t \tag{9}$$

where $\alpha \in (0,1)$. Since loudness decreases geometrically, local searches are closer to the global best as time progresses. For either global or local search, the new position is accepted only if $\beta_3 \sim U(0,1)$ is less than A_i^t and $f(\mathbf{x}_i^{local})$ improves upon $f(\mathbf{x}_*)$. Moreover, a bat adjusts its loudness and rate only if a move is accepted.

Some variants of the bat algorithm use chaotic sequences for parameter initialization (Afrabandpey et al. 2014), Lévy flights for movement (Xie et al. 2013), or chaotic sequences and Lévy flights for movement (Lin et al. 2012).

4.3 Artificial fish swarm (AFS)

The Artificial Fish Swarm Algorithm (Li et al. 2002) is inspired by the behavior of a school of fish searching for food. Fish perform one of four possible actions, namely *move*, *prey*, *swarm*, or *follow*, according to the number of fish in a user-specified visual range (v).

If there are no other fish in visual range, $fish_i$ performs a *move* action, jumping to a random position within its visual range according to

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + s\frac{\mathbf{x}_{dest} - \mathbf{x}_i^t}{||\mathbf{x}_{dest} - \mathbf{x}_i^t||}\beta \tag{10}$$

where $\mathbf{x}_{dest}$ is the fish's destination, s a user-specified step size, and $\beta \sim U(0, 1)$. The quotient results in a unit vector in the direction from $\mathbf{x}_i^t$ to $\mathbf{x}_{dest}$.

If the number of nearby fish exceeds a user-specified crowding threshold (c), a fish performs the *prey* action, where $fish_i$ moves toward $fish_j$ at position $\mathbf{x}_j$, selected uniformly at random from the fish within visible range. The position of the i^{th} fish performing the prey action possesses the same form as Eq. (10), but replaces $\mathbf{x}_{dest}$ with $\mathbf{x}_j$. If $fish_i$ does not *move* or *prey*, it performs the *swarm* action, moving toward the center ($\mathbf{x}_c$) of fish in visual range, if $f(\mathbf{x}_c)$ is better than $f(\mathbf{x}_i)$ according to Eq. (10), with $\mathbf{x}_c$ substituted for $\mathbf{x}_{dest}$. Otherwise, $fish_i$ performs the *prey* action. After swarming or defaulting to preying, if $f(\mathbf{x}_i)$ is worse than $f(\mathbf{x}_*)$, the fish performs the *follow* action, moving toward the current best position $\mathbf{x}_*$ according to Eq. (10), with $\mathbf{x}_*$ in place of $\mathbf{x}_{dest}$.

4.4 Cuckoo search algorithm (CSA)

The Cuckoo Search Algorithm (Yang 2009) is based upon the brood parasitism breeding habits of cuckoos that lay eggs in the nest of other birds. The population is comprised of nests, which are ranked according to their fitness.

In each iteration, a single cuckoo is selected uniformly at randomly and its position updated. The location of the i^{th} cuckoo at time step $(t + 1)$ is

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \alpha \cdot \boldsymbol{L}(\lambda) \tag{11}$$

where the step size vector $\alpha > 0$ is multiplied entry-wise with a vector of Lévy flights (Mantegna 1994)

$$L(\lambda, s) = \frac{\lambda \Gamma(\lambda) \sin(\pi\lambda/2)}{\pi s^{1+\lambda}} \tag{12}$$

where $\lambda \in (1, 3]$ and s is the step size calculated. The new position the cuckoo is compared with an egg in a nest from the population uniformly at random. The position possessing better fitness is preserved and the other eliminated.

Brood parasitism is performed according to the switching parameter $p_a \in [0, 1]$, which determines the percentage of worst solutions eliminated from the population and replaced with a new random solution. At the extremes, $p_a = 0$ can become trapped in a local optimum, whereas $p_a = 1$ reduces to random search.

4.5 Firefly algorithm (FFA)

The Firefly Algorithm (Yang 2009) ascribes individual fireflies an "attractiveness" proportional to its intensity (fitness) and proximity to other fireflies. In each iteration, fireflies move toward other fireflies possessing a higher intensity. These movements are sequential with one firefly making all of its moves before the next firefly. When the i^{th} firefly is attracted to the j^{th} it moves according to

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + I_0 e^{-\gamma r_{ij}^2} (\mathbf{x}_j^t - \mathbf{x}_i^t) + \alpha \boldsymbol{\epsilon} \tag{13}$$

where I_0 is an attraction constant, $\gamma \in (0, 1)$ a user-defined "light-absorption" constant, r_{ij}^2 the Euclidean distance between the i^{th} and j^{th} fireflies, and α a geometrically decreasing scalar according to

$$\alpha = \alpha_0^t \tag{14}$$

with $\alpha_0 \in (0, 1)$ and $\boldsymbol{\epsilon}$ a random vector $\sim U(0, 1)$.

4.6 Flower pollination algorithm (FPA)

The Flower Pollination Algorithm (Yang 2012) is inspired by the pollination process in flowering plants. A user-defined switching probability p_s determines if a particle of pollen from the population performs global or local search.

Global search updates the ith particle's position according to

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \boldsymbol{L}(\boldsymbol{\beta})(\mathbf{g}_* - \mathbf{x}_i^t) \tag{15}$$

where $\mathbf{g}_*$ is the current global best, $\boldsymbol{\beta} \sim \boldsymbol{U}(0, 1)$, and $\boldsymbol{L}$ a step vector drawn from the Lévy distribution given in Eq. (12) with $\lambda \sim U(0, 1)$, Γ the gamma function, and

$$s = \frac{X}{|V|^{1/\lambda}} \tag{16}$$

where $X \sim N(0, \sigma^2)$ and $V \sim N(0, 1)$, with

$$\sigma^2 = \left[\frac{\Gamma(1+\lambda)}{\lambda\Gamma((1+\lambda)/2)} \times \frac{\sin(\pi\lambda/2)}{2^{(\lambda-1)/2}}\right]^{1/\lambda} \tag{17}$$

Local search updates the ith particle's position according to

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \boldsymbol{\beta}(\mathbf{x}_j^t - \mathbf{x}_k^t) \tag{18}$$

where $\boldsymbol{\beta} \sim \boldsymbol{U}(0, 1)$ and particles j and k are selected from the population uniformly at random, restricting the magnitude of a particle's displacement to the distance between the two most distant particles in the population.

4.7 Artificial bee colony (ABC) optimization

The Artificial Bee Colony Algorithm (Karaboga and Basturk 2007) partitions the population into three behaviors, namely scout, experienced, and onlooker bees. The fraction of the population performing these behaviors are user specified parameters.

Scouts perform global search, updating their position according to

$$\mathbf{x}_i^{t+1} = \epsilon(\mathbf{x}_i^t, r) \tag{19}$$

where ϵ is a function that generates a uniformly random point in a sphere of radius r about the present position $\mathbf{x}_i^t$.

Experienced bees update their position in the same manner as PSO updates velocity. Thus, the expression for position updates of experience bees follows the form of Eq. (2) with $\mathbf{x}$ in place of $\mathbf{v}$ and the inertia parameter ω is omitted.

Onlooker bees move a fraction of the distance to a randomly-chosen experienced bee, in order to evaluate new potential solutions in the immediate area, according to the equation

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \beta(\mathbf{p} - \mathbf{x}_i^t) \tag{20}$$

where $\beta \sim U(0, 1)$, and $\mathbf{p}$ is the position of the random experienced bee.

4.8 Wolf search algorithm (WSA)

The Wolf Search Algorithm (Tang et al. 2012) is inspired by the movement patterns of wolves hunting prey and avoiding predators. Each wolf's initial position is uniform and random within the search space and the first step random, updating the position of the i^{th} wolf at time step $(t+1)$ to

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \alpha \cdot r \cdot \boldsymbol{\beta} \tag{21}$$

where $\alpha > 0$ and $r > 0$ are user-defined constants respectively denoting velocity and the wolf's visual range, and $\boldsymbol{\beta} \sim \boldsymbol{U}(0, 1)$.

If there are no other wolves within visual range, a wolf will wander according to Eq. (21). Otherwise, the wolf moves towards the local optimum $(\mathbf{x}_j^t)$ according to

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \boldsymbol{\beta}_0 e^{-R^2}(\mathbf{x}_j^t - \mathbf{x}_i^t) + \mathbf{M} \tag{22}$$

where $\boldsymbol{\beta}_0$ is the current global best position, R the Euclidean distance between $\mathbf{x}_i^t$ and $\boldsymbol{\beta}_0$, and $\mathbf{M}$ is a random position within the visual range. The term $\boldsymbol{\beta_0} e^{-R^2}$ exhibits the inverse square law. Thus, as the distance between the two wolves increases, attractiveness decreases, similar to the attractiveness factor in the Firefly Algorithm.

The probability of a wolf encountering a predator is $p_a \in (0, 1)$. If such cases, a wolf retreats to a position outside of its visual range according to

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \alpha \cdot s \cdot \mathbf{M} \tag{23}$$

where s is a user-defined step constant and $\mathbf{M}$ calculates a uniform random position outside of the visual range but inside the global search space. Thus, predators encourage diversity within the search. At extremes, $p_a = 0$ introduces no randomization, while $p_a = 1$ is random.

5 Likelihood functions

This section describes the likelihood or objective functions to which the multi-phase algorithm design framework is applied. Functions considered include nonhomogeneous Poisson process (Farr 1996) and covariate (Shibata et al. 2006) software reliability growth models.

5.1 Nonhomogeneous Poisson process software reliability growth model

This section describes nonhomogeneous Poisson process software reliability growth models. 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 (ANSI/IEEE 1991).

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

$$LL(t_i; \Theta) = -m(t_n) + \sum_{i=1}^{n} \log\left(\lambda(t_i)\right) \tag{24}$$

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 SRGMn (Yamada and Osaki 1983) is

$$m(t) = a\left(1 - e^{-bt^c}\right) \tag{25}$$

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 Fig. 3. 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 (Okamura et al. 2003), 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 (Goel 1985) model and $s = 2$ is a user-defined parameter to control the width of the interval.

5.2 Covariate software reliability growth model

This section describes a NHPP software reliability growth model possessing a discrete Cox proportional hazards rate (DCPH) incorporating covariates (Nagaraju et al. 2020), which correspond to multiple test activities to expose defects. Numerical estimates of these covariates quantify the effectiveness of each test activity.

Let Y_i be the number of defects discovered in time interval i such that $S_n = \sum_{i=1}^{n} Y_i$ is the total number of defects discovered through the first n intervals. Furthermore, vector $\beta = (\beta_1, \beta_2, \ldots, \beta_j)$ denotes j covariates (software test activities) and $\mathbf{x}_i = (x_1, x_2, \ldots, x_j)$ the amount of time dedicated to each test activity to expose software defects in the ith interval. The log-likelihood function of the NHPP software reliability growth model incorporating covariates is

$$\begin{aligned} LL(\theta, \beta, \omega) = &-\omega \sum_{i=1}^{n} p_{i,\mathbf{x}_i:\theta,\beta} + \sum_{i=1}^{n} y_i \ln(\omega) \\ &+ \sum_{i=1}^{n} y_i \ln(p_{i,\mathbf{x}_i:\theta,\beta}) - \sum_{i=1}^{n} \ln(y_i!). \end{aligned}$$

where ω is the mean of the Poisson distribution, θ is the vector of model parameters, and $p_{i,\mathbf{x}_i:\theta,\beta}$ is the Cox proportional hazard function given by

$$p_{i,\mathbf{x}_i:\theta,\beta} = \left(1 - (1 - h_{i:\theta}^0)^{g(\mathbf{x}_i,\beta)}\right) \prod_{k=1}^{i-1} (1 - h_{k:\theta}^0)^{g(\mathbf{x}_k,\beta)} \tag{26}$$

with $g(\mathbf{x}_i, \beta) = \exp(\mathbf{x}_i, \beta)$ and $h_{i:\theta}^0$ is the baseline hazard function such as the geometric distribution

$$h_{i;b}^0 = b \tag{27}$$

and $b \in (0, 1)$.

Only a single method to generate initial parameter estimates for the covariate SRGM was employed because closed form expressions for initial parameters based on the expectation

Table 2 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

maximization algorithm are difficult to obtain and the computational complexity of applying this approach numerically is also prohibitive. Therefore, alternative initial parameter estimation strategies described in Fig. 3 were not implemented. Instead, the single approach used for this stage was uniform random numbers $b \sim U(0.8, 0.99)$ and $\beta \sim U(0, 0.1)$ for each covariate in the data set.

6 Illustrations

This section illustrates the application of the framework to nonhomogeneous Poisson process and covariate SRGM. The NSGA-II and range of swarm algorithm parameters are first described, followed by design and cross-validation experiments and accompanying discussion.

Table 2 summarizes the parameters of NSGA-II, including the number of generations, population size, precision of numerical parameters, and crossover logic.

The values for the number of generations and population were selected to enable more extensive search. Specifically, a larger number of generations allowed for improvement within a promising design, while a larger population allowed for competition between designs incorporating different "species" of swarm algorithms. It was observed that smaller populations tended to allow one swarm algorithm to drive competition to extinction prematurely, even if the swarm algorithm found in the majority of the population did not demonstrate the desired properties of rapid and consistent convergence. The number of runs was chosen to be an odd value because the fitness of designs was determined as the median in order to avoid optimistic or pessimistic outliers that would have suggested performance was faster or slower or that convergence was better or worse due to randomness. As described in the illustrations, experiments based on 31 runs exhibited some variability, which may have moderately degraded accurate assessment of a design and its position on the Pareto curve relative to competing designs. Therefore, 63 runs were performed in all subsequent experiments and the fitness of runtime and accuracy computed according to the median in order to avoid being mislead by favorable or unfavorable outcomes in the tails of the distributions of these runs. The number of bits per parameter was chosen to be 32 because this allowed each numerical parameter to be represented to several decimal points precision. For example, a parameter in the interval $(0, 1)$ could assume values in steps of $\frac{1}{32^2-1} = 2.3283110^{-10}$. The crossover probability was chosen to be high because this promotes greater diversity during search, as opposed to simply allowing two parents to progress unaltered to the next generation without attempts to achieve further improvement. Head-tail crossover was employed because it is the most common form of creating two offspring from two parents and tends to preserve some degree of consistency from generation to generation.

Table 3 Swarm algorithm parameter ranges

Algorithm	Param	Description	θ^-	θ^+
PSO	ω	Velocity weight	0.35	0.65
	ϕ_p	Single best weight	0.05	0.15
	ϕ_g	Swarm best weight	0.05	0.15
Bat	f_{min}	Minimum movement frequency	0.00	0.20
	f_{max}	Maximum movement frequency	0.80	1.00
	α	Loudness scale	0.82	0.98
	γ	Pulse control	0.65	0.95
Fish		Visual scope	0.05	0.20
		Crowd size	2	6
	s	Step size	0.05	0.2
		Behavior iterations	1	3
Cuckoo	λ	Lévy lambda	0.94	1.00
	α	Lévy scale	0.94	1.00
	p_a	Abandon percentage	0.15	0.35
Firefly	γ	Light absorption	0.90	1.00
	α	Randomness scalar	0.92	1.00
Flower	p_s	Switching probability	0.50	0.90
Bee	p_o	Onlooker percentage	0.23	0.43
	p_e	Experienced percentage	0.23	0.43
	w_b	Single best weight	0.25	0.75
	w_g	Swarm best weight	0.25	0.75
	r	Radius multiplier	0.10	0.30
Wolf	r	Search radius	0.40	0.60
	α	Step multiplier	0.70	1.00
	s	Global multiplier	0.40	0.60
	p_a	Global search percentage	0.825	0.975

Table 3 shows the range of values considered for the parameters of each swarm algorithm. While the full interval is (0, 1) in many cases, many algorithms perform best when some parameters are close to zero or one because they exhibit poor convergence or performance otherwise. Moreover, experiments confirmed that optimal designs were within the sub-intervals specified, suggesting that the search was not overly constrained. In nearly all cases, the interval was selected to vary around a commonly recommended point value (Yang 2014). For example, $\omega = 0.5 \pm 0.15$ in PSO. Wider intervals are possible. However, this may unnecessarily degrade the performance of swarm algorithms. Inspection of NSGA-II during execution also indicated that the best performing designs resided within the interior of these intervals.

6.1 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 Eqs. (24) and (25) on the SYS1

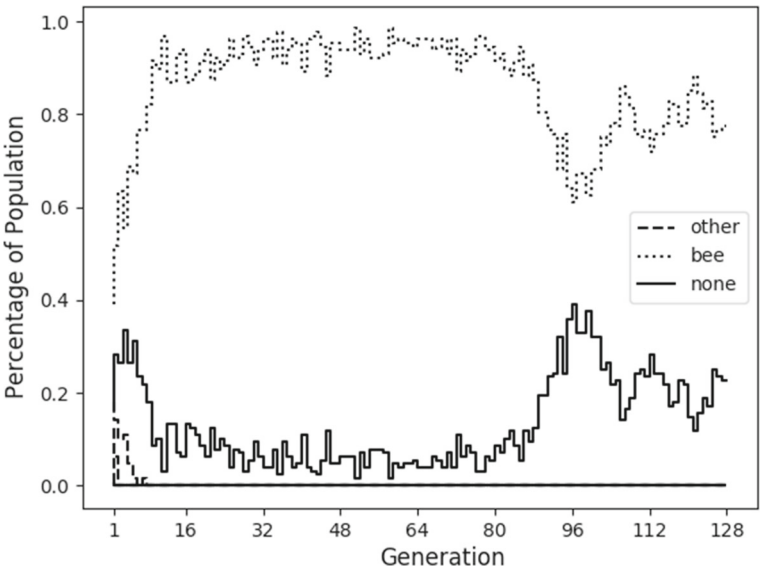

Fig. 4 Percentage of population utilizing alternative swarm algorithms in each generation of NSGA-II

data set (Lyu 1996). 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 4 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 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 Eq. (24) on the SYS1 data (Lyu 1996) 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.

To compare alternative designs, Fig. 5 shows a Pareto plot composed of the 128 members of the population in the final generation. Figure 5 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 s) 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.

Solutions at the knee of the curve in the bottom left of Fig. 5 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 s 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 s. Thus,

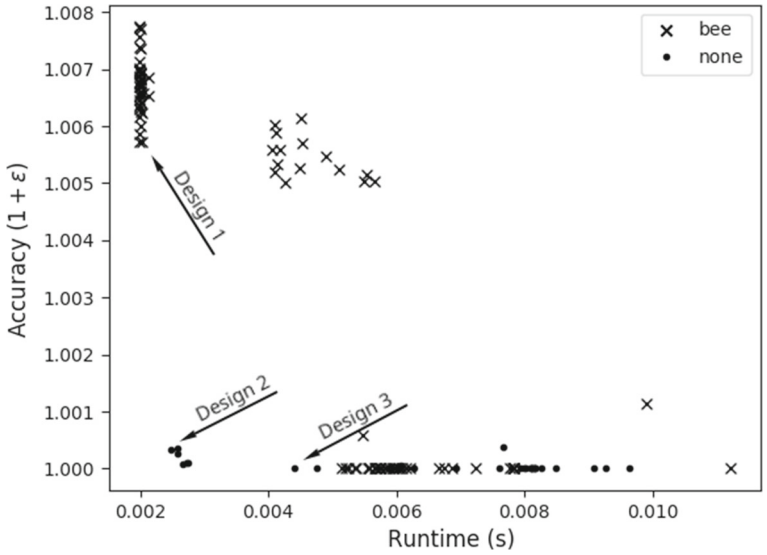

Fig. 5 Pareto frontier of multi-phase algorithm designs in final generation of NSGA-II on Weibull NHPP SRGM

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 Fig. 5.

To test the generalizability of our designs, we ran the two unique designs identified in Fig. 5 on eight additional failure times data sets from the software reliability literature (Lyu 1996), 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 6 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, size 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 s, similar to the range (0.005, 0.010) observed in Fig. 5. 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.

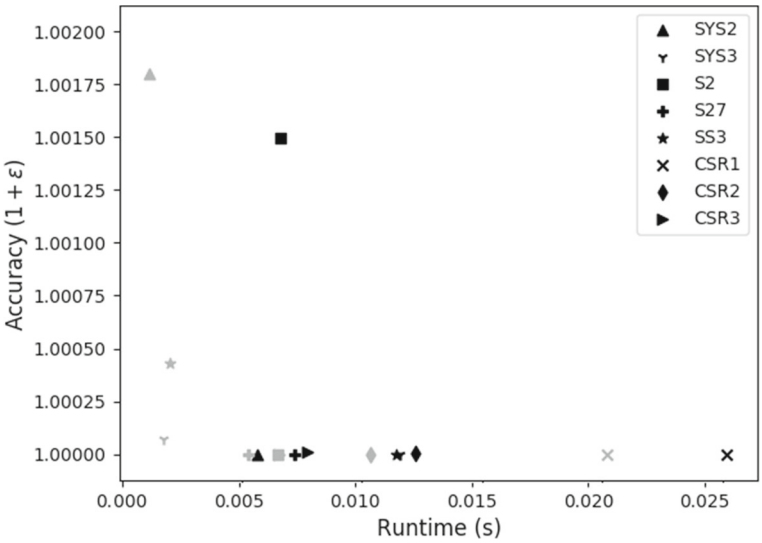

Fig. 6 Cross-validation Pareto frontier

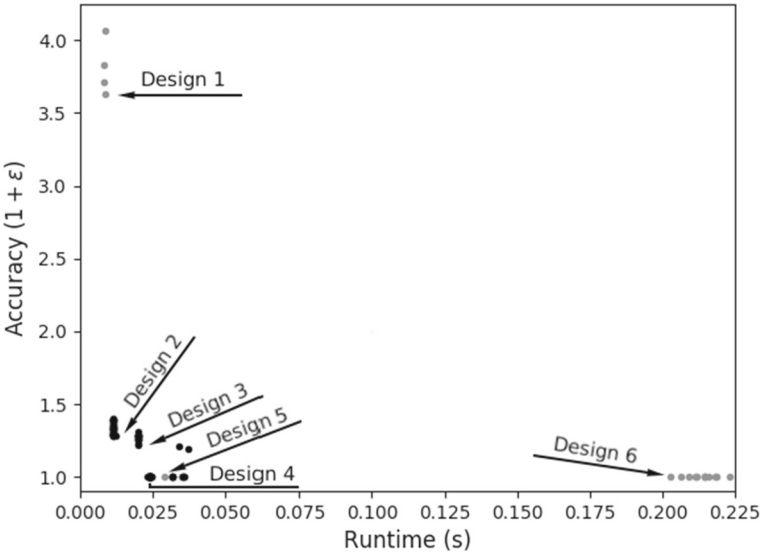

Fig. 7 Pareto frontier of multi-phase algorithm designs in final generation of NSGA-II on covariate NHPP SRGM

6.2 Covariate software reliability growth model

Since multi-phase algorithms incorporating swarm algorithms did not exhibit a decisive improvement over traditional numerical methods on a nonhomogeneous Poisson process SRGM, this example conducted a similar set of experiments on a software reliability model including three covariates. This design stage required about 3.5 hours to complete.

Figure 7 shows six designs identified with data set DS2 (Shibata et al. 2006).

Table 4 Bat design parameters

Design	Gen	Pop	f_{min}	f_{max}	α	γ
2	6	6	0.021768	0.917212	0.825154	0.823620
3	14	6	0.021768	0.922107	0.825076	0.823620
4	6	6	0.046744	0.922789	0.825152	0.755835

Six designs, not three, were selected because the first three designs identified contained a swarm stage. Therefore, to enable comparison, three additional designs without a swarm stage were selected from along the Pareto frontier. Specifically, Designs 2-4 correspond to multi-phase algorithms consisting of both a swarm and numerical stage. In all three cases, the swarm stage consisted of a bat algorithm with parameters reported in Table 4. Design 2 and 3 utilized the limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm and Design 4 the sequential least squares procedure. Designs 1, 5, and 6 were the best alternatives composed of just a numerical stage implementing (i) the limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm, (ii) the sequential least squares procedure, and (iii) Powell's method respectively. Of these, Designs 4 and 5 were the two most competitive alternatives because they both exhibited high median accuracy and low median run time.

The cross-validation step applied Designs 4 and 5 to data set DS1 (Shibata et al. 2006), requiring only a few minutes. Figures 8 and 9 respectively show the accuracy and run time of the designs over 1000 runs. Figure 8 indicates that 99.5% of runs of Design 4 attained convergence, with error $\varepsilon < 10^{-6}$, whereas Design 5 converged in only 62.4% of the runs. Moreover, Fig. 9 indicates that the corresponding empirical upper 99.5% confidence limit on the run time of Design 4 was less than 0.17 s, whereas only 43.5% of the runs of Design 5 completed in this amount of time, exhibiting an average run time nearly three times as large. One possible explanation for this discrepancy is that Design 4 included swarm and numerical stages, but Design 5 only included a numerical stage, which may have impeded convergence in some runs. Thus, Design 4 performed best with respect to both run time an accuracy. Moreover, to eliminate the 0.05% non-convergence in Design 4, further experimentation and fine tuning can modestly increase the number of iterations of the bat algorithm to take advantage of its global search properties.

7 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 nonhomogeneous Poisson process and covariate software reliability growth models. Cross-validation was performed on other failure time and covariate data sets. The results suggested that the NHPP SRGM considered did not benefit significantly from a multi-phase algorithm. However, a multi-phase algorithm incorporating the bat algorithm exhibited notably higher performance and stability than the next best alternative, only employed a numerical method. Thus, the proposed framework can be used to identify algorithmic designs suitable for the complexity of a model. The present paper there-

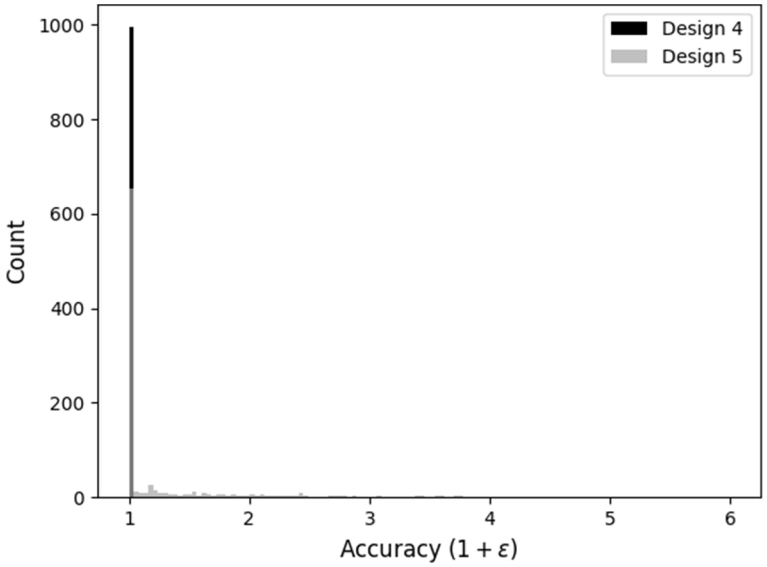

Fig. 8 Accuracy of design 4 and 5 during cross-validation

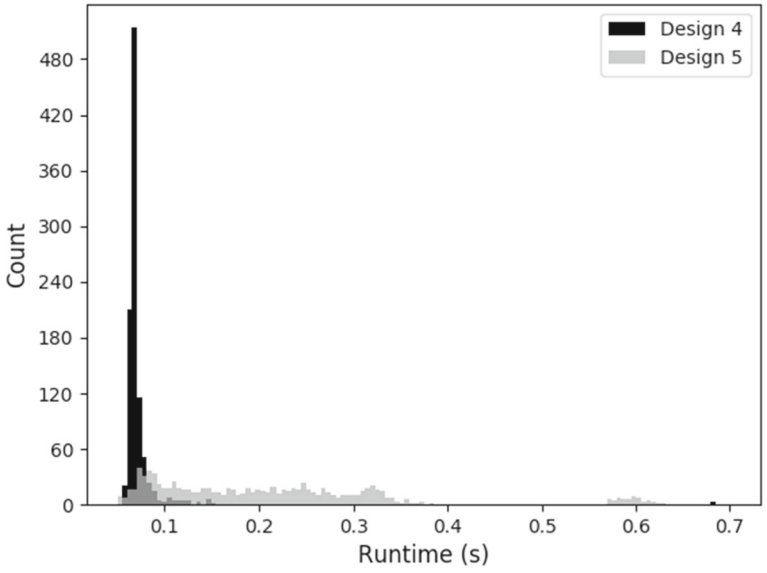

Fig. 9 Run time of design 4 and 5 during cross validation

fore enables the application of covariate SRGM to higher dimensional data in a manner that is both stable and efficient. The practical contribution of these stable and efficient algorithms is their potential for implementation in a tool to promote the adoption of covariate models by non-experts in the software reliability and software security testing communities.

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.

Possible extensions of the framework include alternatives to NSGA-II to balance the performance and stability of algorithmic designs as well as additional statistical cross-validation methods. The framework is also capable of accommodating additional swarm algorithm variants and numerical methods. Additional efforts to scale the approach to covariate SRGM with many software metrics or a comprehensive set of software security testing techniques will also be pursued. More generally, the framework should find application to a variety of non software reliability optimization problems, promoting novel collaborations between modeling and algorithmic researchers from diverse areas of interest.

Author Contributions Conceptualization: LF; Methodology: JS, JA, KC, VN, LF; Formal analysis and investigation: JS and LF; Writing - original draft preparation: JS, JA, KC, VN, LF; Writing - review and editing: JS and LF; Funding acquisition: LF; Resources: LF; Supervision: LF.

Funding This work was supported by National Science Foundation Award (#1749635).

Availability of data and material Not applicable.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Code availability GitHub.

References

Afrabandpey, H., Ghaffari, M., Mirzaei, A., & Safayani, M. (2014). A novel bat algorithm based on chaos for optimization tasks. In *Proc. Iranian conference on intelligent systems* (pp. 1–6).

Alabajee, M., & Alreffaee, T. (2018). Exploring ant lion optimization algorithm to enhance the choice of an appropriate software reliability growth model. *International Journal of Computer Applications, 182*(4), 1–8.

Alneamy, J., & Dabdoob, M. (2017). The use of original and hybrid grey wolf optimizer in estimating the parameters of software reliability growth models. *International Journal of Computer Applications, 167*(3), 12–21.

Al-Saati, D., & Abd-AlKareem, M. (2013). The use of cuckoo search in estimating the parameters of software reliability growth models. *International Journal of Computer Science and Information Security, 11,* 22.

Al-Saati, N., & Alabajee, M. (2016). On the performance of firefly algorithm in software reliability modeling. *Proceedings International Journal of Recent Research and Review, 9*(4), 1–9.

Altaf, I., Majeed, I., & Arshid Iqbal, K. (2016). Effective and optimized software reliability prediction using harmony search algorithm. In *Proc. international conference on green engineering and technologies* (pp. 1–6).

ANSI/IEEE. (1991). Standard glossary of software engineering terminology (std-729-1991). Tech. rep., ANSI/IEEE.

Archetti, F., & Schoen, F. (1984). A survey on the global optimization problem: General theory and computational approaches. *Annals of Operations Research, 1*(2), 87–110.

Bansal, J., Singh, P., Saraswat, M., Verma, A., Jadon, S., & Abraham, A. (2011). Inertia weight strategies in particle swarm optimization. In *Proc. world congress on nature and biologically inspired computing* (pp. 633–640).

Bertsekas, D. (1999). Nonlinear programming. *Athena, 48,* 334.

Bonnans, J. F., Gilbert, J. C., Lemaréchal, C., & Sagastizábal, C. A. (2006). *Numerical optimization theoretical and practical aspects*. Springer.

Byrd, R. H., Lu, P., Nocedal, J., & Zhu, C. (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing, 16*(5), 1190–1208.

Chen, M., Wu, H., & Shyur, H. (2001). Analyzing software reliability growth model with imperfect-debugging and changepoint by genetic algorithms. In *Proc. international conference on computers and industrial engineering* (pp. 520–526).

Choudhary, A., Baghel, A. S., & Sangwan, O. P. (2018). Parameter estimation of software reliability model using firefly optimization. In *Proc. data engineering and intelligent computing* (pp. 407–415). Springer.
Choudhary, A., Baghel, A. S., & Sangwan, O. P. (2017). An efficient parameter estimation of software reliability growth models using gravitational search algorithm. *International Journal of System Assurance Engineering and Management, 8*(1), 79–88.
Choudhary, A., Baghel, A. S., & Sangwan, O. P. (2017). Efficient parameter estimation of software reliability growth models using harmony search. *IET Software, 11*(6), 286–291.
Costa, E. O., Vergilio, S. R., Pozo, A. T. R., & Souza, G. A. (2005). Modeling software reliability growth with genetic programming. In *Proceedings international symposium on software reliability engineering*.
Costa, E. O., de Souza, G. A., Pozo, A. T. R., & Vergilio, S. R. (2007). Exploring genetic programming and boosting techniques to model software reliability. *IEEE Transactions on Reliability, 56*(3), 422–434.
Dai, Y., Xie, M., Poh, K., & Yang, B. (2003). Optimal testing resource allocation with genetic algorithm for modular software systems. *Journal of Systems and Software, 66*(1), 47–55.
Deb, K. (2015). *Multi-objective evolutionary algorithms* (pp. 995–1015). Springer. https://doi.org/10.1007/978-3-662-43505-2_49
Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6*(2), 182–197. https://doi.org/10.1109/4235.996017
Dohi, T., Nishio, Y., & Osaki, S. (1999). Optimal software release scheduling based on artificial neural networks. *Annals of Software Engineering, 8,* 167–185.
Eberhart, R., & Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Proc. IEEE international symposium on micro machine and human science* (pp. 39–43).
Farr, W. (1996). Software reliability modeling survey. In M. Lyu (Ed.), *Handbook of software reliability engineering* (vol. 222, pp. 71–117). McGraw-Hill .
Fletcher, R. (1986). *Practical methods of optimization*. Wiley.
Goel, A. (1982). *Software reliability modelling and estimation techniques*. Tech. rep., Department of Industrial Engineering and Operations Research, Syracuse University.
Goel, A. (1985). Software reliability models: Assumptions, limitations, and applicability. *IEEE Transactions on Software Engineering, SE–11*(12), 1411–1423. https://doi.org/10.1109/tse.1985.232177
Goldberg, D. E. (2012). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley.
Grippo, L., Lampariello, F., & Lucidi, L. (1989). A truncated newton method with nonmonotone line search for unconstrained optimization. *Journal of Optimization Theory and Applications, 60*(3), 401–419.
Hassanien, A., & Emary, E. (2016). *Swarm intelligence: Principles, advances, and applications*. CRC Press.
Hestenes, M., & Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards, 49*(6), 409.
Hudaib, A., & Moshref, M. (2018). Survey in software reliability growth models: Parameter estimation and models ranking. *International Journal of Computer Systems, 5*(5), 11–25.
Jin, C., & Jin, S. W. (2016). Parameter optimization of software reliability growth model with s-shaped testing-effort function using improved swarm intelligent optimization. *Applied Soft Computing, 40,* 283–291.
Karaboga, D., & Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. *Journal of Global Optimization, 39*(3), 459–471.
Karunanithi, N., Whitley, D., & Malaiya, Y. K. (1992). Prediction of software reliability using connectionist models. *IEEE Transactions on Software Engineering, 18*(7), 563–574.
Kaswan, K., Choudhary, S., & Sharma, K. (2015). Software reliability modeling using soft computing techniques: Critical review. *Journal of Information Technology and Software Engineering, 5,* 144.
Kumar, A., Tripathi, R.P., Saraswat, P., & Gupta, P. (2017). Parameter estimation of software reliability growth models using hybrid genetic algorithm. In *Proc. IEEE international conference on image information processing* (pp. 1–6).
Lin, J., Chou, C., Yang, C., & Tsai, H. L. (2012). A chaotic Levy flight bat algorithm for parameter estimation in nonlinear dynamic biological systems. *Computer and Information Technology, 2*(2), 56–63.
Li, X. L., Shao, Z. J., & Qian, J. X. (2002). An optimizing method based on autonomous animats: Fish-swarm algorithm. *Systems Engineering—Theory & Practice, 22*(11), 32.
Li, Z., Yu, M., Wang, D., & Wei, H. (2019). Using hybrid algorithm to estimate and predicate based on software reliability model. *IEEE Access, 7,* 84268–84283.
Lu, K., & Ma, Z. (2018). 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* (pp. 268–271).
Lyu, M. (Ed.). (1996). *Handbook of software reliability engineering*. McGraw-Hill.
Lyu, M., & Nikora, A. (1992). CASRE—A computer-aided software reliability estimation tool. In *Proc. of computer-aided software engineering workshop* (pp. 264–275).

Mantegna, R. N. (1994). Fast, accurate algorithm for numerical simulation of Lévy stable stochastic processes. *Physical Review E, 49*(5), 4677–4683. https://doi.org/10.1103/physreve.49.4677

Messac, A., Ismail-Yahaya, A., & Mattson, C. (2003). The normalized normal constraint method for generating the Pareto frontier. *Structural and Multidisciplinary Optimization, 25*(2), 86–98.

Minohara, T., & Tohma, Y. (1995). Parameter estimation of hyper-geometric distribution software reliability growth model by genetic algorithms. In *Proc. international symposium on software reliability engineering* (pp. 324–329).

Mohanty, R., Ravi, V., & Patra, M. R. (2010). The application of intelligent and soft-computing techniques to software engineering problems: A review. *International Journal of Information and Decision Sciences, 2*(3), 233–272.

Mueller-Gritschneder, D., Graeb, H., & Schlichtmann, U. (2009). A successive approach to compute the bounded pareto front of practical multiobjective optimization problems. *SIAM Journal on Optimization, 20*(2), 915–934.

Nagaraju, V., Fiondella, L., Zeephongsekul, P., Jayasinghe, C., & Wandji, T. (2017). Performance optimized expectation conditional maximization algorithms for nonhomogeneous Poisson process software reliability models. *IEEE Transactions on Reliability, 66*(3), 722–734.

Nagaraju, V., Jayasinghe, C., & Fiondella, L. (2020). Optimal test activity allocation for covariate software reliability models. *Journals of Systems and Software, 168,* 110643.

Nelder, J. A., & Mead, R. (1965). A simplex method for function minimization. *The Computer Journal, 7*(4), 308–313.

Okamura, H., Watanabe, Y., & Dohi, T. (2002). Estimating mixed software reliability models based on the EM algorithm. In *Proceedings international symposium on empirical software engineering* (pp. 69–78).

Okamura, H., Watanabe, Y., & Dohi, T. (2003). An iterative scheme for maximum likelihood estimation in software reliability modeling. In *Proc. international symposium on software reliability engineering* (pp. 246–256).

Pai, P. F., & Hong, W. C. (2006). Software reliability forecasting by support vector machines with simulated annealing algorithms. *Journal of Systems and Software, 79,* 747–755.

Powell, M. J. D. (1964). An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal, 7*(2), 155–162.

Rao, M., & Anuradha, K. (2016). 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* (pp. 1–8).

Shanmugam, L., & Florence, L. (2012). A comparison of parameter best estimation method for software reliability models. *International Journal of Software Engineering and Applications, 3,* 91–102.

Sharma, D.T., Pant, M., & Abraham, A. (2011). 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* (pp. 207–212).

Sheta, A. (2006). Reliability growth modeling for software fault detection using particle swarm optimization. In *Proc. IEEE congress on evolutionary computation* (pp. 3071–3078).

Sheta, A., & Abdel-Raouf, A. (2016). Estimating the parameters of software reliability growth models using the grey wolf optimization algorithm. *International Journal of Advanced Computer Science and Applications, 7*(4), 499–505.

Shibata, K., Rinsaka, K., & Dohi, T. (2006). Metrics-based software reliability models using non-homogeneous Poisson processes. In *Proc. international symposium on software reliability engineering* (pp. 52–61).

Shibata, K., Rinsaka, K., & Dohi, T. (2015). M-srat: Metrics-based software reliability assessment tool. *International Journal of Performability Engineering, 11*(4), 369–379.

Tang, R., Fong, S., Yang, X., & Deb, S. (2012). Wolf search algorithm with ephemeral memory. In *Proc. international conference on digital information management* (pp. 165–172). https://doi.org/10.1109/ICDIM.2012.6360147

Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Jarrod Millman, K., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C., Polat, İ., Feng, Y., Moore, E. W., Van der Plas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., & Contributors SciPy1.0, C. (2019). SciPy 1.0–fundamental algorithms for scientific computing in python. arXiv e-prints arXiv:1907.10121

Xie, J., Zhou, Y., & Chen, H. (2013). A novel bat algorithm based on differential operator and Lévy flights trajectory. *Computational Intelligence and Neuroscience, 2013*.

Xing, F., & Guo, P. (2005). Support vector regression for software reliability growth modeling and prediction. In *Proc. advances in neural networks* (pp. 925–930).

Yamada, S., & Osaki, S. (1983). Reliability growth models for hardware and software systems based on nonhomogeneous Poisson processes: A survey. *Microelectronics Reliability, 23*(1), 91–112.
Yang, X. S. (2009). Firefly algorithms for multimodal optimization. In *International symposium on stochastic algorithms* (pp. 169–178). Springer.
Yang, X. (2009). Suash Deb: Cuckoo search via Lévy flights. In *Proc. world congress on nature biologically inspired computing* (pp. 210–214). https://doi.org/10.1109/NABIC.2009.5393690
Yang, X. S. (2010). A new metaheuristic bat-inspired algorithm, vol. nature inspired cooperative strategies for optimization (pp. 65–74). Springer.
Yang, X. S. (2012). Flower pollination algorithm for global optimization. In *Proc. international conference on unconventional computing and natural computation* (pp. 240–249). Springer.
Yang, X. S. (2014). *Nature-inspired metaheuristic algorithms*. Elsevier.
Zeephongsekul, P., Jayasinghe, C., Fiondella, L., & Nagaraju, V. (2016). Maximum-likelihood estimation of parameters of NHPP software reliability models using expectation conditional maximization algorithm. *IEEE Transactions on Reliability, 65*(3), 1571–1583.
Zheng, C., Liu, X., Huang, S., & Yao, Y. (2011). A parameter estimation method for software reliability models. *Procedia Engineering, 15,* 3477–3481.
Zitzler, E., Laumanns, M., & Thiele, L. (2001). SPEA 2: Improving the strength pareto evolutionary algorithm. *TIK-Report, 103*.