A HYBRID MODEL FITTING APPROACH INCORPORATING PARTICLE SWARM OPTIMIZATION AND STATISTICAL ALGORITHMS*

VIDHYASHREE NAGARAJU
Tandy School of Computer Science, University of Tulsa, OK, USA
email: vidhyashree-nagaraju@utulsa.edu

LANCE FIONDELLA†
Department of Electrical and Computer Engineering
University of Massachusetts Dartmouth, MA, USA
email: lfiondella@umassd.edu

This chapter implements experimental methods that combine the global search properties of particle swarm optimization (PSO) to obtain a near optimal solution, followed by a numerical or statistical method to obtain a precise optimum. Runtime is explicitly considered in order to identify the number of iterations and PSO population that converge consistently while minimizing overall runtime. Experiments are performed in the context of a non-homogeneous Poisson process (NHPP) software reliability growth model (SRGM).

1. Introduction

Software reliability growth models [1] are fit to time series data associated with failures experienced during testing in order to predict measures such as the time required to achieve a desired failure intensity or time between failures. Historically, numerical algorithms such as Newton's method were employed, which required good initial parameter estimates and therefore a high-level of expertise to apply SRGM. Recent approaches to overcome the instability of classical numerical methods include techniques such as swarm intelligence [2], which exhibits robust global search. However, these techniques can require significant computing resources and time to converge to a precise optimum, which is important for SRGM because some model parameters are very sensitive to accurate estimates of other parameters. Moreover, most past research applying swarm intelligence

*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.
†Corresponding Author

techniques do not consistently consider the runtime required, yet both stability and performance are important, especially when an algorithm is to be implemented in a tool so that nonexperts can apply SRGM quickly and confidently without needing to learn the underlying mathematics. To achieve this pragmatic goal, more comprehensive methods are needed to assess the consistency of convergence as well as the runtime of these algorithms.

Among existing swarm intelligence methods, particle swarm optimization [3] has been identified as a stable and efficient algorithm [4] to identify maximum likelihood estimates of SRGM [2]. The first application of PSO to NHPP SRGM [2] was motivated by the efficiency of PSO on related software defect prediction problems. Subsequent studies applying PSO to identify the MLEs of traditional NHPP SRGM include [5]. Due to the slower runtime of PSO on more complex models, studies have also compared PSO runtime with alternative algorithms [6].

Numerical and statistical methods for software reliability model fitting include the expectation-maximization (EM) algorithm [7] and expectation conditional maximization (ECM) algorithm [8,9]. When initial parameter estimates are far from the maximum, the EM and ECM algorithms are stable, but can exhibit slow convergence. To overcome these limitations, this paper implements experimental methods that combine the global search properties of PSO to obtain a near optimal solution, followed by a numerical or statistical methods to obtain a precise optimum. Runtime is explicitly considered in order to identify the number of iterations and PSO population that converge consistently while minimizing overall runtime. Experiments are performed in the context of a NHPP SRGM [10]. Our results indicate that a hybrid approach converges more rapidly than either alternative in isolation.

The remainder of the paper is organized as follows: Section 2 reviews software reliability growth models, while Section 3 describes parameter estimation methods. Section 4 presents a sequence of illustrative examples demonstrating how particle swarm optimization is combined with traditional methods in a manner to conduct rigorous tradeoff assessment between convergence and runtime. Section 5 offers conclusions and suggests future research.

2. Nonhomogeneous Poisson Process Software Reliability Growth Models

This section describes nonhomogeneous Poisson process software reliability growth models. The NHPP [11] counts the number of events that occur by time t. In the context of SRGM, this counting process $N(t)$ corresponds to the number

of unique software defects detected by time t. The expected value or average of the counting process is denoted $m(t) := E[N(t)]$. For example, the mean value function of the Weibull SRGM [10] is

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

where a represents the number of faults that would be discovered if the testing was continued indefinitely, while b and c are the scale and shape parameters respectively. Many models may be expressed in the general form $a \times F(t)$, where $F(t)$ is a cumulative distribution function (CDF).

The instantaneous failure rate of a SRGM is $\lambda(t) := \frac{\partial m(t)}{\partial t}$. Therefore, the instantaneous failure rate of the Weibull SRGM is

$$\lambda(t) = abct^{c-1}e^{-bt^c} \tag{2}$$

Given vector of failure times $\mathbf{T} = \langle t_1, t_2, \ldots, t_n \rangle$, fitting a model requires the maximization of the log-likelihood function.

$$LL(\theta, a) = -m(t_n) + \sum_{i=1}^{n} \log\left(\lambda(t_i)\right) \tag{3}$$

3. Parameter Estimation Algorithms

This section discusses techniques to establish initial parameter estimates based on the expectation maximization algorithm as well as a strategy to combine this technique with particle swarm optimization. Next, methods to precisely identify the maximum likelihood estimates of model parameters, including the expectation conditional maximization algorithm are discussed.

3.1. *Initial Parameter Estimates*

The EM algorithm [7] provides a calculus-based method to obtain initial estimates of a model. The initial estimate of parameter a is

$$a^{(0)} = n \tag{4}$$

The remaining parameters of the distribution function $F(\bullet;\theta)$ are computed as

$$\theta^{(0)} := \sum_{i=1}^{N} \frac{\partial}{\partial \theta} \log\left[f(;\theta)\right] = \mathbf{0} \tag{5}$$

where $\mathbf{0}$ is the zero vector of length $|\theta|$, the number of parameters to be estimated.

Differentiating the portion of Equation (1) that corresponds to $F(t)$ produces $f(t)$, which is substituted into Equation (5) to obtain,

$$b^{(0)} = \frac{n}{\sum_{i=1}^{n} t_i^c}, \tag{6}$$

and $c^{(0)} = 1$, where c has been set to the feasible initial value one because Equation (5) lacks a closed form expression, but $c^{(0)} = 1$ simplifies to the special case of the exponential model. These initial estimates can be used with the ECM algorithm, or any other strategy such as particle swarm optimization to maximize the log likelihood function. For example, we generate initial positions of particles uniformly at random in $\left(\frac{1}{\alpha}\theta_i^{(0)}, \alpha\theta_i^{(0)}\right)$, where $\alpha > 1$ is a scaling parameter.

3.2. *Particle Swarm Optimization (PSO)*

Particle swarm optimization [3] is a computational method to iteratively search an m-dimensional space for the global optimum of an objective function $f(\mathbf{x})$ such as a log-likelihood function. Toward this end, PSO maintains a population of $|p|$ candidate solutions (particles), each of which possesses a position and velocity. The velocity vector of the i^{th} particle at discrete time step $(t+1)$ is

$$\mathbf{v}_i^{t+1} = a\mathbf{v}_i^t + c_1 v_1(\mathbf{p}_i - \mathbf{x}_i^t) + c_2 v_2(\mathbf{g} - \mathbf{x}_i^t) \tag{7}$$

where $\mathbf{v}_i^t$ is the velocity vector of the particle i at time step t, $\mathbf{p}_i$ is the best position visited by the i^{th} particle in all time steps up to and including the present time so that $f(\mathbf{p}_i) \geq f(\mathbf{p}_i^t)$, while $\mathbf{g}$ is the global best position visited by any particle within the population. Since $\mathbf{x}_i^t$ denotes the position of the i^{th} particle at time step t, $(\mathbf{p}_i - \mathbf{x}_i^t)$ and $(\mathbf{g} - \mathbf{x}_i^t)$ are the vectors pointing from a particle's present position to the direction of the particle and global best respectively.

3.3. *Expectation Conditional Maximization (ECM) Algorithm*

The expectation conditional maximization algorithm [8,9] simplifies maximum likelihood estimation by dividing a single M-step of the EM algorithm [9] into $|\theta|$ conditional-maximization (CM) steps. EM algorithms apply $|\theta|$ update rules in a single M-step, whereas the CM steps of the ECM algorithm update one parameter at a time holding the $|\theta| - 1$ other parameters constant at their present values. This reduces maximum likelihood estimation to a sequence of $|\theta|$ one-dimensional problems.

The CM-steps of the Weibull software reliability growth model are [8]

$$b'' = \frac{n}{\sum_{i=1}^{n} t_i^{c'} - \frac{n t_n^{c'}}{1 - e^{b'' t_n^{c'}}}} \tag{8}$$

$$c'' = \frac{-n}{\sum_{i=1}^{n} \left(\log(t_i)(1 - b' t_i^{c''})\right) + \frac{n b' t_n^{c''} \log(t_n)}{1 - e^{b' t_n^{c''}}}} \tag{9}$$

4. Illustrations

This section examines several alternative combinations of algorithms to fit traditional NHPP software reliability growth models. We first conduct a detailed analysis in the context of particle swarm optimization, studying the tradeoff between iterations and population size. In light of this analysis, the runtime performance and convergence of model fitting algorithms including and excluding PSO are compared. Three combinations are considered and each is identified with a unique numeric value referred to as a Design for the sake of clarity.

All designs utilize the EM algorithm to obtain initial estimates because good initial estimates often accelerate convergence significantly. Designs I and II represent cases where a single method is employed, including PSO and the expectation conditional maximization algorithm respectively. Design III employ PSO followed by the expectation conditional maximization algorithm to assess if the global search properties of PSO coupled with a numerical method lowers the time to converge to a near optimal solution.

4.1. *PSO Tradeoff Analysis*

This example assesses the tradeoff between the number of iterations and number of particles in the swarm.

The examples are performed in the context of the Weibull software reliability growth model. Specifically, the PSO algorithm is applied to maximize the log-likelihood function given in Equation (3), after substituting Equations (1) and (2) for the mean value function and failure intensity respectively. The SYS1 failure times data set [1], which consists of $n = 136$ failures is then substituted in for t_i. The PSO parameters (a, c_1, and c_2) were set to 0.5 and the coefficient for initial positions (α) to 2.0 was applied to determine initial estimates $a^{(0)} = 136$, $b^{(0)} = 4.04 \times 10^{-5}$, and $c^{(0)} = 1.0$ from Equations (4)-(6) of the EM algorithm.

4.1.1. *Constant Number of Function Evaluations*

This section explores the tradeoff between the number of iterations and population size when the number of function evaluations is held constant at $1,024$.

Figure 1 shows the average log-likelihood attained after a specified number of iterations for populations ranging from 1 to 256, where averages in this and subsequent experiments were computed from 100 independent runs in order to illustrate trends more clearly. The x-axis shows the number of iterations on an exponential scale because smaller populations ran for a greater number of iterations such that $p \times i = 1024$. For example, the lower curve shows the average for

the special case possessing a population of one, in which the particle and global best are the same. Due to the degenerate size of the population, the single particle is unable to benefit from sharing across particles in the swarm and was therefore only able to reach an average log-likelihood value of -973. Similarly, the line above shows the progress made by a population of two particles over 512 iterations. Thus, the right endpoints of the curves for a population of size one and size two correspond to a total of $1,024$ function evaluations. The population of two ends on average approximately two points higher, suggesting that increasing the population and decreasing the number of iterations produces a better result, despite approximately equal work overall. The curves above these two show the average performance of populations between 4, and 256, decreasing the number of iterations in order to hold the number of function evaluations constant.

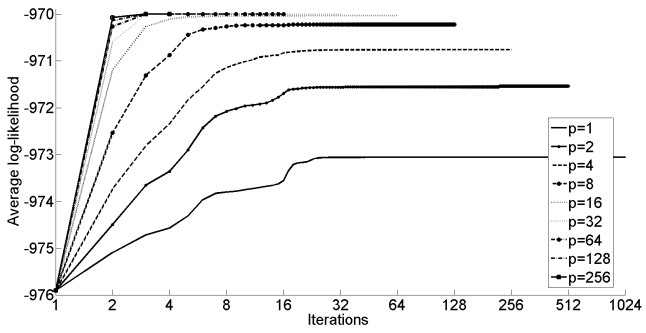

Figure 1.: Tradeoff between iterations and population on average log-likelihood for fixed number of function evaluations (Design I)

Figure 1 also indicates that the marginal utility between a population of two and four, measured as the difference in log-likelihood at the right endpoints, is smaller than the increase when the population is doubled from one to two. Moreover, the marginal utility decreases as the population increases from 2^i to 2^{i+1}, approaching 0 as the population increases from 16 to 32. Moving left along the curves that achieve a near optimum (populations of 16 and greater), we see that it took approximately eight iterations for the population of 16 to reach an average log-likelihood of approximately -970, whereas a population of 32 took approximately 4 iterations. Thus, approximately 128 function evaluations were sufficient to reach a near optimal value, indicating that $7/8$ths of the function evaluations contributed very little to convergence, but consumed nearly 90% of the compu-

tational power and processing time. Moreover, none of these combinations is sufficient to consistently achieve the maximum likelihood value of -966.0803, confirming that PSO may be a good global search method, but that classical optimization techniques must also be employed to reach the maximum.

4.2. *PSO+ECM performance*

Figure 2 shows the average runtime of Design III. The solid vertical line corresponds to the average time required by the ECM algorithm (Design II).

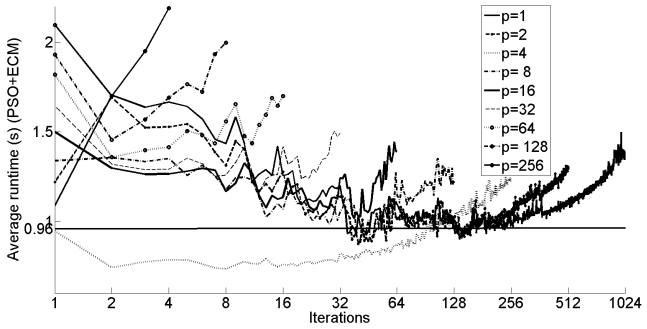

Figure 2.: Average runtime of PSO+ECM (Design IV)

Figure 2 illustrates that most combinations of PSO iterations and population sizes required more time than Design II, which does not employ a PSO stage. However, a population of $p = 4$ particles and eight iterations of PSO followed by the expectation maximization algorithm exhibited an average runtime of 0.7345 seconds, representing a speedup factor of 1.3 ($0.9581/0.7345$) over Design II. Figure 2 demonstrates that hybrid designs composed of PSO plus a traditional algorithm can achieve speedup, but that sensible application of PSO must take the additional runtime into consideration.

5. Conclusion and Future Work

This chapter combined the global search properties of PSO to obtain a near optimal solution, followed by a statistical method to obtain a precise optimum. Runtime was considered to identify the number of iterations and PSO population that converged consistently while minimizing overall runtime. Our results indicate that a hybrid approach can converge more rapidly than either alternative in isolation

and that the approach may be more suitable for models with additional parameters, which is a subject of future research.

Bibliography

[1] M. Lyu, editor. Handbook of Software Reliability Engineering. McGraw-Hill, New York, NY, 1996.

[2] A. Sheta. Reliability growth modeling for software fault detection using particle swarm optimization. *IEEE International Conference on Evolutionary Computation*, pages 3071–3078, 2006.

[3] J. Kennedy and R R. Eberhart. Particle swarm optimization (PSO). *IEEE International Conference on Neural Networks*, Australia, pages 1942–1948, 1995.

[4] M. Bonyadi and Z. Michalewicz. Analysis of stability, local convergence, and transformation sensitivity of a variant of the particle swarm optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 20(3):370–385, 2015.

[5] I. Altaf, F. Rashid, J. Dar, and M. Rafiq. Survey on parameter estimation in software reliability. *IEEE International Conference on Soft Computing Techniques and Implementations*, pages 22–27, 2015.

[6] C. Zheng, X. Liu, S. Huang, and Y. Yao. A parameter estimation method for software reliability models. *Procedia Engineering*, 15:3477–3481, 2011.

[7] H. Okamura, Y. Watanabe, and T. Dohi. An iterative scheme for maximum likelihood estimation in software reliability modeling. *IEEE International Symposium on Software Reliability Engineering*, pages 246–256, 2003.

[8] 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*, 66(3):722–734, 2017.

[9] 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*, 65(3):1571–1583, 2016.

[10] S. Yamada and S. Osaki. Reliability growth models for hardware and software systems based on nonhomogeneous Poisson processes: A survey. *Microelectronics Reliability*, 23(1):91–112, 1983.

[11] S. Ross. Introduction to Probability Models. *Academic Press*, New York, NY, 8th edition, 2003.