ChinaRxiv

AI translation · View original & related papers at
chinarxiv.org/items/chinaxiv-201808.00127

Dynamic Reliability Allocation for Complex Software Systems Based on Differential Evolution: Postprint

Authors: Zhang Shijin, Zhang Guofu, Su Zhaopin, Yue Feng

Date: 2018-07-23T00:00:00+00:00

Abstract

Existing research on reliability allocation for complex software systems is predicated on software systems with fixed architectures; however, in practical scenarios, software system architectures are often not fixed. To resolve this contradiction, a dynamic reliability allocation optimization model for complex software systems is constructed, and a dynamic reliability allocation algorithm for complex software systems is designed based on differential evolution. When the system architecture changes, the global weights of each module in the system are first re-evaluated based on D-S evidence theory, and considering the correlation between the system before and after the change, partial historical solutions are retained when generating the initial population in differential evolution. Finally, the effectiveness of the proposed method is verified through simulation experiments and analysis.

Full Text

Dynamic Reliability Allocation of Complex Software Systems Based on Differential Evolution

Authors: Zhang Shijin[1], Zhang Guofu[1,2,3], Su Zhaopin[1,2,3], Yue Feng[1,2,3]
Affiliations:
1. School of Computer & Information, Hefei University of Technology, Hefei 230601, China
2. Anhui Province Key Laboratory of Industry Safety & Emergency Technology, Hefei 230601, China
3. Engineering Research Center of Safety Critical Industrial Measurement & Control Technology for Ministry of Education, Hefei 230601, China

Abstract: Existing research on reliability allocation for complex software systems assumes a fixed system structure, whereas in practice software system

structures frequently change. To address this discrepancy, this paper constructs a dynamic reliability allocation optimization model for complex software systems and designs a dynamic reliability allocation algorithm based on differential evolution. When the system structure changes, we first re-evaluate the global weights of each module using Dempster-Shafer evidence theory. Considering the correlation between the system before and after the change, we retain some historical solutions when generating the initial population for differential evolution. Finally, simulation experiments verify the effectiveness of the proposed method.

Keywords: complex software system; dynamic reliability allocation; differential evolution; Dempster-Shafer evidence theory; reserve historical solutions

0 Introduction

Software reliability design has always been a crucial and active research area in software engineering. With the widespread deployment of large-scale complex software systems in safety-critical domains such as power systems, railway transportation, aerospace, and national security, the demand for software system reliability has grown increasingly stringent, drawing greater attention from researchers and designers to software reliability allocation problems [**?**, **?**, **?**]. However, software reliability cannot be measured precisely; it can only be assessed through system testing. For software development engineers, achieving optimal reliability through software reliability allocation during the early development stages represents a vital component of software engineering [**?**].

In reliability allocation modeling, Zahedi et al. [**?**] constructed a software system reliability maximization model with cost constraints. Leung [**?**, **?**] defined software reliability functions based on operational profiles, considering each customer' s software usage patterns while satisfying practical constraints such as cost budgets and software quality requirements. Kapur et al. [**?**] combined system redundancy with budget constraints to maximize system reliability and also considered compatibility issues among alternatives available for different modules. Roy et al. [**?**] designed a software reliability allocation scheme for multifunctional multi-user digital relay systems to improve the reliability of transmission line fault detection, classification, and localization. Chatterjee et al. [**?**] proposed a software reliability allocation model incorporating user perspectives, establishing a system hierarchy to integrate user viewpoints with those of software managers and programmers, thereby reflecting both user requirements and programmer recommendations in technical design and reliability considerations.

Regarding solution methods for reliability allocation problems, Han et al. [**?**] formulated software system reliability allocation as a constrained combinatorial optimization problem based on simulated annealing genetic algorithms, employing simulated annealing fitness scaling methods and multi-point crossover to improve solution quality. Xu et al. [**?**] evaluated reliability for complex software

with sequential, concurrent, loop, and fault-tolerant architectures using a hybrid intelligent optimization algorithm based on penalty functions, distribution estimation, and adaptive crossover differential evolution [?] (DE). Sangeetha et al. [?] utilized multi-objective genetic algorithms to solve multi-objective reliability allocation problems considering reliability, cost, and schedule. However, these existing works focus solely on single software system structures, whereas complex software systems often comprise multiple software components. Moreover, most previous research employs analytic hierarchy processes to predict parameter values for each module, failing to account for uncertainty and incompleteness in practical engineering applications. To address this, Yue et al. [?] constructed a cost-constrained system reliability maximization model based on complex software system hierarchies, employed Dempster-Shafer evidence theory [?] to estimate parameter values for each module, and used differential evolution to search for optimal reliability allocation schemes.

Nevertheless, all aforementioned works consider static scenarios for software reliability allocation. It is important to note that during actual software development, system structures are often not fixed due to changing user requirements or programmer improvements. Reliability allocation schemes established before such changes clearly cannot adapt to the modified system structure, while completely re-optimizing new allocation schemes based on the current structure proves time-consuming and labor-intensive. To address this scenario, this paper first proposes a dynamic reliability allocation model for complex software systems that simulates reliability allocation problems under changing system structures. We solve this dynamic optimization problem based on Dempster-Shafer evidence theory and DE algorithms, while considering historical solutions during optimization to improve final solution quality. Simulation experiments finally verify the effectiveness of the proposed approach.

---

1 Problem Description

The hierarchical structure of complex software systems typically follows a top-down approach, as illustrated in [Figure 1: see original paper]. The first layer is the software layer, representing users' overall assessment of each software' s reliability. The second layer is the function layer, where each software contains multiple functions. The third layer is the program layer, where each function is implemented by several programs. The fourth layer is the module layer, where each program comprises multiple modules. Here, each module is independent, and reliability indices are allocated only to the module layer.

Typically, the reliability degree of users performing various functions in software is measured by "software utility."In this hierarchical structure, the program layer reflects both users' system views and programmers' recommendations regarding software reliability, making it common to study the utility of complex software systems from the program layer perspective.

The utility of program P_i is calculated as:

$$U_i = w_i^P \cdot r_i^P$$

where w_i^P is the global weight of program P_i and r_i^P is the reliability of program P_i. Since modules are independent units, r_i^P can be calculated as:

$$r_i^P = \prod_{j \in M_i} r_j^M$$

where r_j^M is the reliability allocated to module M_j. Additionally, each module M_j has a global weight w_j^M.

The dynamic reliability allocation problem for complex software systems involves allocating reliability to each module in the system under certain budget constraints to maximize the system' s overall utility. The optimization model is formulated as:

$$\begin{aligned}
\max \quad & U = \sum_{i=1}^{n} w_i^P \cdot r_i^P = \sum_{i=1}^{n} w_i^P \cdot \prod_{j \in M_i} r_j^M \\
\text{s.t.} \quad & 0 \le r_j^M \le 1, \quad j = 1, 2, \ldots, m \\
& \sum_{j=1}^{m} (a_j + b_j \cdot r_j^M) \le B_l, \quad l = 1, 2, \ldots, p \\
& \sum_{j \in S_l} (a_j + b_j \cdot r_j^M) \le B_l, \quad l = 1, 2, \ldots, p
\end{aligned}$$

In this model, the value of m is dynamic, meaning the number of modules changes. The objective function U represents the overall utility of the entire complex software system as a function of each module' s reliability r_j^M. Since the value of m changes dynamically, the variable set $\{r_j^M\}$ is also dynamic. Constraint (4) bounds module reliability between 0 and 1. Constraint (5) represents the budget constraint for module costs, where a_j is the fixed overhead cost for module M_j to achieve reliability r_j^M, b_j is the variable cost (the additional cost per unit increase in reliability), B_l is the budget for software S_l, and $r_j^{M_{\max}}$ is the maximum reliability for module M_j. Constraint (6) represents the cost constraint for software S_l.

---

2 Dynamic Reliability Allocation Algorithm

2.1 Differential Evolution

Differential evolution (DE) is a population-based heuristic search algorithm [**?**]. The main evolutionary process includes mutation, crossover, and selection.

The DE algorithm implements individual mutation through differential strategies. A common differential strategy randomly selects two different individuals from the population, scales their vector difference, and combines it with the original individual to generate a new mutant individual V_i:

$$V_i = X_{r1} + F \cdot (X_{r2} - X_{r3})$$

where X_{r1} and X_{r2} are two randomly selected individuals different from X_i, and F is the scaling factor.

The crossover operation aims to produce a new trial individual U_i by exchanging gene values between the original individual X_i and the mutant individual V_i based on crossover probability CR. If a random number $\text{rand}(0,1) \leq CR$, then $U_i = V_i$; otherwise, $U_i = X_i$. The trial individual U_i is then evaluated using the fitness function, and superior individuals are selected to form the next generation. If the fitness value of U_i is better than X_i, then $X_i = U_i$; otherwise, X_i remains unchanged.

2.2 Dempster-Shafer Evidence Theory

Dempster-Shafer evidence theory [**?**] combines multiple pieces of evidence to make decisions and achieve a certain degree of belief through reasonable interpretation of reasoning. Yue et al. [**?**] employed Dempster-Shafer evidence theory to estimate the global weights of each program and module. This paper adopts the same approach to re-estimate the global weights of each program and module in the changed system structure.

Let Θ be the sample space of possible values for variable X, called a frame of discernment. The basic probability assignment function $\mu : 2^{\Theta} \to [0,1]$ satisfies $\mu(\emptyset) = 0$ and $\sum_{A \subseteq \Theta} \mu(A) = 1$. Here, $\mu(A)$ represents the precise belief degree in A, known as the basic probability number of set A.

When data from different sources provide different assessment information for the same frame of discernment, the following combination rule is used for information fusion. If B and C are two different pieces of evidence for the same frame of discernment, then:

$$\mu(A) = \frac{\sum_{B \cap C = A} \mu_1(B) \cdot \mu_2(C)}{1 - K}, \quad A \neq \emptyset$$

where $K = \sum_{B \cap C = \emptyset} \mu_1(B) \cdot \mu_2(C)$ is the normalization constant that calculates the sum of basic probabilities for all non-empty sets, preventing non-zero probability from being assigned to the empty set during combination. This effectively represents the degree of conflict between evidence, with larger values indicating greater conflict.

In this paper, we first determine each software' s global weight w_i^P based on its budget B_l. Users and programmers then provide local weights for each function, program, and module according to their importance within their respective

software, function, and program. These local weights appear as values on each line in [Figure 1: see original paper]. Using the calculation and combination method from [**?**], we derive the global weights w_i^P and w_j^M for each program and module across the entire complex software system. Notably, when the number of modules changes, we only need to estimate each module' s new global weight w_j^M based on the unchanged global weights of each program and the new local weights of each module.

2.3 Algorithm Description

As shown in [Figure 2: see original paper], during the optimization process, if the system structure changes, we first use Dempster-Shafer evidence theory to estimate each module' s new global weight based on its new local weights. We then use the DE algorithm to search for the optimal reliability allocation scheme under the new system structure, with Equation (3) as the fitness function and Equations (4)-(6) as constraints. Specifically, during population initialization, we sort individuals by fitness value and retain the best 75% of individuals from historical searches as part of the initial solution for current optimization, while randomly initializing the remaining 25% of individuals according to constraints.

---

3 Simulation Experiments and Analysis

3.1 Experimental Environment

To verify the effectiveness of the proposed method, we designed a complex software system containing three states (S_1, S_2, and S_3) during optimization. The system hierarchy and local weights of each unit in the three states are shown in [Figure 3: see original paper]. The budgets for S_1, S_2, and S_3 are 250, 450, and 300, respectively.

State S_2 differs from S_1 in that program P_2 adds a new module M_8, and program P_5 adds a module M_9. The local weights of modules associated with P_2, P_5, and P_4 change accordingly, while the local weights of modules in P_1 and P_3 remain unchanged. State S_3 differs from S_2 in that program P_1 adds a module M_{10}, program P_3 adds two modules M_{11} and M_{12}, and program P_4 adds a module M_{13}. The local weights of modules associated with P_1, P_3, and P_4 change, while the local weights of modules in P_2 and P_5 remain unchanged.

In the DE algorithm, the crossover probability $CR = 0.5$, scaling factor $F = 0.94$, and population size is 60.

3.2 Experimental Results

Using Dempster-Shafer evidence theory, the global weights at the program layer are $\{0.2628, 0.0284, 0.5106, 0.0582, 0.14\}$. The global weights at the module layer for the three states are shown in .

** Global Weights at Module Layer for Each State**

[Figure content describing table structure would appear here]

[Figure 4: see original paper] shows the optimization results when DE transitions from state S_1 to S_2 after 10, 20, and 30 iterations in state S_1. "Pure static optimization" refers to DE searching directly in state S_2; "Considering historical solutions" refers to DE using partial historical solutions from state S_1 as the initial population when the structure changes to S_2; "Not considering historical solutions" refers to DE randomly generating the initial population when the structure changes to S_2. Both "considering" and "not considering" historical solutions belong to dynamic optimization. [Figure 4: see original paper] demonstrates that considering historical solutions converges significantly faster than not considering them, indicating that retaining historical solutions from the previous state can shorten the distance between the initial population and the optimal population. Moreover, with sufficient iterations in state S_1 to ensure adequate exploration by DE, dynamic optimization can find solutions similar to pure static optimization even with fewer iterations in state S_2.

[Figure 5: see original paper] presents the optimization results when DE operates in state S_1 for 5 iterations, then state S_2 for 5 iterations, followed by state S_2 for 10 iterations and state S_3 for 10 iterations, and finally state S_2 for 15 iterations and state S_3 for 15 iterations. The results show that as the frequency of changes increases, the gap between dynamic and static optimization becomes more pronounced, indicating that frequent system structure changes hinder the optimization of reliability allocation problems. However, considering historical solutions consistently outperforms not considering them. This advantage is particularly evident when the total number of iterations is limited but iterations in earlier states are substantial, as the final state lacks sufficient iterations for adequate DE exploration. Therefore, in challenging dynamic optimization scenarios, considering historical solutions proves to be an effective means of improving final solution quality.

---

4 Conclusion

Addressing the reality that complex software system structures may change over time due to evolving user requirements or programmer improvements during reliability allocation optimization, this paper constructs a dynamic reliability allocation optimization model for complex software systems and designs a corresponding algorithm based on Dempster-Shafer evidence theory and differential evolution. When the system structure changes, we first re-estimate the local weights of each module using Dempster-Shafer evidence theory and retain partial historical solutions when generating the initial population for differential evolution. Simulation results verify the effectiveness of the proposed method. However, it should be noted that when the number of modules increases significantly, the objective function may become unsolvable. The second group of

experiments also indicates that as the number of modules grows, the solution space of the objective function becomes very small, which severely limits the performance of differential evolution. Therefore, the proposed algorithm is only suitable for scenarios with a relatively small number of modules. Future work should consider: dynamic multi-objective reliability allocation problems; optimization problems when changes occur at the software, function, or program layers; and designing constraint handling techniques based on the upper and lower bounds of U to improve algorithm search performance.

References

[1] Xu Renzuo, Zhang Liangpin, Zhang Dashuai. A nonlinear programming model of software reliability allocation [J]. Computer Engineering, 2003, 29(17): 34-36.

[2] Gao feng, Wu Lingbo, Yue Yang, et al. Software reliability combination model based on AHP and AdaBoosting [J]. Computer Engineering, 2017, 43(12): 69-72.

[3] Zhang Ce, Meng Fanchao, Kao Yonggui, et al. Survey of software reliability growth model [J]. Journal of Software, 2017, 28(9): 2402-2430.

[4] Zahedi F, Ashrafi N. Software reliability allocation based on structure, utility, price, and cost [J]. IEEE Trans on Software Engineering, 1991, 17(4): 345-356.

[5] Leung Y W. Optimal reliability allocation for modular software system designed for multiple customers [J]. IEICE Trans on Information & Systems, 1996, 79(12): 1655-1662.

[6] Leung Y W. Software reliability allocation under an uncertain operational profile [J]. Journal of the Operational Research Society, 1997, 48(4): [page numbers].

[7] Kapur P K, Bardhan A K, Jha P C. Optimal Reliability Allocation Problem for a Modular Software System [J]. Opsearch, 2003, 40(2): 138-148.

[8] Roy D S, Mohanta D K, Panda A K. Software reliability allocation of digital relay for transmission line protection using a combined system hierarchy and fault tree approach [J]. IET Software, 2008, 2(5): 437-445.

[9] Chatterjee S, Singh J B, Roy A. A structure-based software reliability allocation using fuzzy analytic hierarchy process [J]. International Journal of Systems Science, 2015, 46(3): 513-525.

[10] Han Bingqing, Gao Jianhua. Reliability allocation and research of software system based on Simulated Annealing Genetic Algorithm [J]. Computer Engineering, 2003, 29(4): 67-69.

[11] Xu Yue, Pi Dechang. Complex software reliability allocation based on hybrid intelligent optimization algorithm [J]. Journal of Software, 2018, 29(9): 1-19.

[12] Das S, Mullick S S, Suganthan P N. Recent advances in differential evolution: an updated survey [J]. Swarm & Evolutionary Computation, 2016, 27: 1-30.

[13] Sangeetha M, Arumugam C, Kumar K M S, et al. An effective approach to support multi-objective optimization in software reliability allocation for improving quality [J]. Procedia Computer Science, 2015, 47: 118-127.

[14] Yue F, Zhang G, Su Z, et al. Multi-software reliability allocation in multimedia systems with budget constraints using Dempster–Shafer theory and improved differential evolution [J]. Neurocomputing, 2015, 169: [page numbers].

[15] Yager R R, Liu L. Classic Works of the Dempster-Shafer Theory of Belief Functions [M]// Classic Works on the Dempster-Shafer Theory of Belief Functions (Studies in Fuzziness and Soft Computing). New York: Springer-Verlag, 2007: 1-34.

Note: Figure translations are in progress. See original paper for figures.

Source: ChinaXiv —Machine translation. Verify with original.